Détecteur de présence Bluetooth

Diplômant/e Jimmy Dubuis

Objectif du projet

Le but de ce projet est de concevoir une unité de contrôle utilisant la technologie Bluetooth Low Energy afin d'autoriser ou non l'alimentation d'une lampe ou d'un actuateur par la seule présence d'un téléphone mobile.

Méthodes | Expériences | Résultats

Dans le cadre de ce projet, le Bluetooth Low Energy est utilisé. Ce protocole permet une faible consommation et utilise peu d'échanges de données. De ce fait, l'impact énergétique est bien inférieur à un détecteur de présence infrarouge.

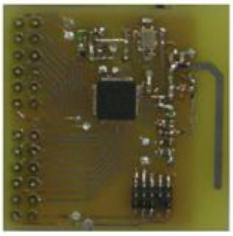
Une étude de cas d'utilisation a été réalisée avec comme premier cas la/les lampe/s comme maître des transactions et comme deuxième cas le/s téléphone/s comme maître des transactions. Ceci a permis de définir les services à utiliser.

Ensuite, la conception de l'unité de contrôle est réalisée autour du composant nRF51822. Cette unité de contrôle permet de communiquer par Bluetooth Low Energy et d'effectuer des mesures de tension.

Cette unité de contrôle a ensuite été utilisée pour piloter trois démonstrateurs :

- Une lampe solaire bon marché achetée sur le marché
- Une lampe solaire développée lors de ce projet autour du composant bq25504
- Une prise murale développée lors de ce projet composée de deux prises électriques de 230V

Les différentes étapes préparatoires avant la phase d'industrialisation ont été menées avec succès. L'utilisation d'application déjà existante pour iPhone a permis de valider les différents cas d'utilisation. Une application propriétaire reste cependant à développer.



Unité de contrôle développée lors de ce projet

Démonstrateur prise murale

Travail de diplôme
| édition 2013 |

Filière
Systèmes industriels

Domaine d'application
Infotronics

Professeur responsable
Pierre Pompili
pierre.pompili@hevs.ch

Partenaire
Cottet Electronic SA

TABLE DES MATIERES

1 INTRODUCTION

De nos jours, les lampes de jardin nécessitent de l'énergie solaire pour éclairer. Si l'on veut augmenter la durée de fonctionnement de ces lampes, il suffit d'éclairer seulement si quelqu'un se trouve à proximité de la lampe. Pour se faire, des capteurs de présence existent. La consommation de courant de ce type de capteur s'amplifie dès que la distance de détection augmente.

L'arrivée de la technologie Bluetooth Low Energy dans les téléphones mobiles offre une nouvelle forme de détection de présence, performante et peu gourmande en énergie. Il serait donc possible de détecter, non plus la personne, mais son téléphone mobile.

L'objectif de ce projet est de concevoir une unité de contrôle utilisant la technologie Bluetooth Low Energy pour autoriser ou non l'alimentation d'une lampe solaire ou d'un actuateur par la seule présence d'un téléphone mobile.

2 OBJECTIFS

En premier lieu, une étude UML sur les cas d'utilisation de la détection Bluetooth Low Energy sera réalisée. Cette étude permettra de spécifier les paramètres de communication entre le téléphone mobile et le module Bluetooth. Ces paramètres permettront de programmer le module Bluetooth Low Energy.

Ensuite, une board contenant uniquement les composants permettant le fonctionnement du module Bluetooth a été développé. Cette board pourra être utilisée avec trois variantes différentes :

- Variante adaptable à une lampe solaire : dans cette variante, la board du module Bluetooth est adaptée à une lampe solaire bon marché achetée dans le commerce.
- Variante lampe solaire : dans cette variante, une lampe solaire est entièrement développée.
- Variante prise murale : Dans cette variante, un système de commutation intégrable dans une prise murale a été développé. Ceci, afin de pouvoir par exemple, commander des stores depuis un téléphone mobile.

3 CAS D'UTILISATION

3.1 INTRODUCTION

Avec le Bluetooth Low Energy, deux rôles principaux sont utilisés : l'advertiser et le scanner. L'advertiser va émettre périodiquement une trame afin d'indiquer à un scanner qui se trouve à proximité son adresse Bluetooth. Une fois que le scanner reçoit cette adresse, il a la possibilité de se connecter à l'advertiser.

Le scanner a la particularité de pouvoir mesurer le RSSI (Received Signal Strength Indication) du signal de l'advertiser. Le RSSI va varier en fonction de la puissance du signal reçu et donc en fonction de la distance à laquelle se trouve l'advertiser du scanner.

Dans le cadre de ce projet, deux scénarios sont possibles. En effet, le module Bluetooth peut être advertiser ou scanner. Ce chapitre décrit le fonctionnement des deux cas d'utilisation.

3.2 SCENARIO : LAMPE SOLAIRE ADVERTISER

Dans ce chapitre, la lampe solaire est l'*advertiser* et l'*iPhone* est le *scanner*.

3.2.1 1 lampe et 1 iPhone

Principe:

La lampe va s'annoncer en envoyant son adresse Bluetooth chaque 2 secondes. L'iPhone qui se situe à proximité va recevoir ce signal, mesurer le RSSI et définir si la lampe doit être allumée ou éteinte. Si elle doit être allumée, l'iPhone va se connecter, puis se déconnecter à la lampe solaire. Chaque fois qu'un iPhone se connectera à la lampe, cette dernière s'allumera.

Si aucun iPhone ne s'est connecté durant les 4 secondes qui suivent la dernière connexion, la lampe s'éteint. A chaque nouvelle connexion d'un iPhone, la lampe va rester allumée et attendra à nouveau 4 secondes pour s'éteindre.

La figure 1 présente le use case diagramme et la figure 2 le diagramme de séquence.

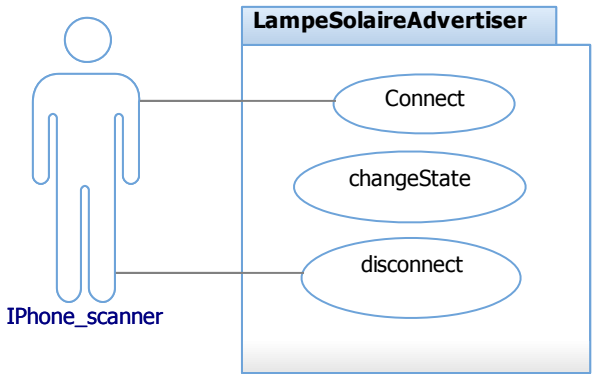


Figure 1 : Lampe adv. : 1 lampe et 1 iPhone: use case diagram

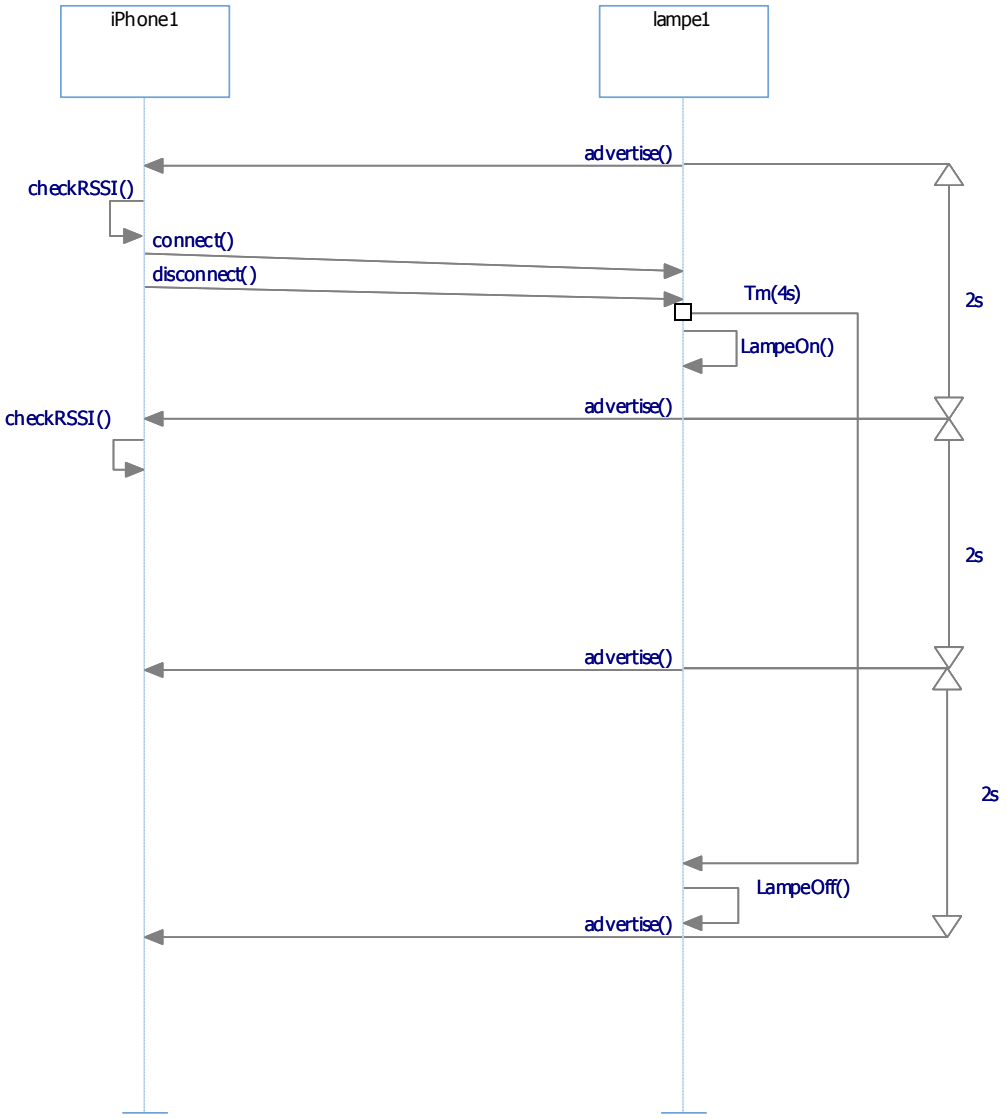


Figure 2 : Lampe adv. : 1 lampe et 1 iPhone: sequence diagram

3.2.2 3 lampes et 1 iPhone

Principe :

Dans ce scénario, il y a trois lampes et un iPhone. Les lampes vont s'annoncer à l'iPhone. Puis l'iPhone va se connecter aux trois lampes pour les allumer. Le cas présenté sur le diagramme de séquence de la figure 4 a de très faible chance d'arriver. En effet, les trois lampes ne s'annonceront généralement pas en même temps. Ce cas particulier fonctionne correctement, car l'annonce se fait sur trois canaux différents. Un canal étant une bande de fréquences différentes. Le cas des trois lampes qui n'émettent pas en même temps fonctionne également correctement.

Si le nombre de lampes avait été de 4 à s'annoncer en même temps, une des lampes n'aurait pas été détectée. Mais ce cas de figure a très peu de chance d'arriver.

La figure 3 présente le use case diagramme et la figure 4 le diagramme de séquence de ce scenario.

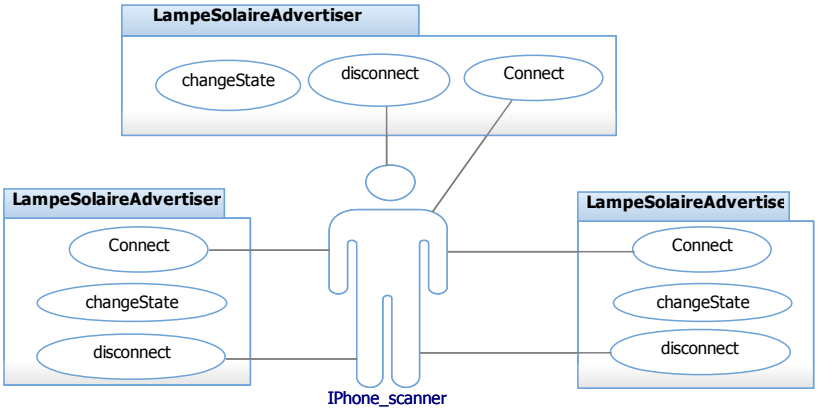


Figure 3 : Lampe adv. : 3 lampes et 1 iPhone : use case diagram

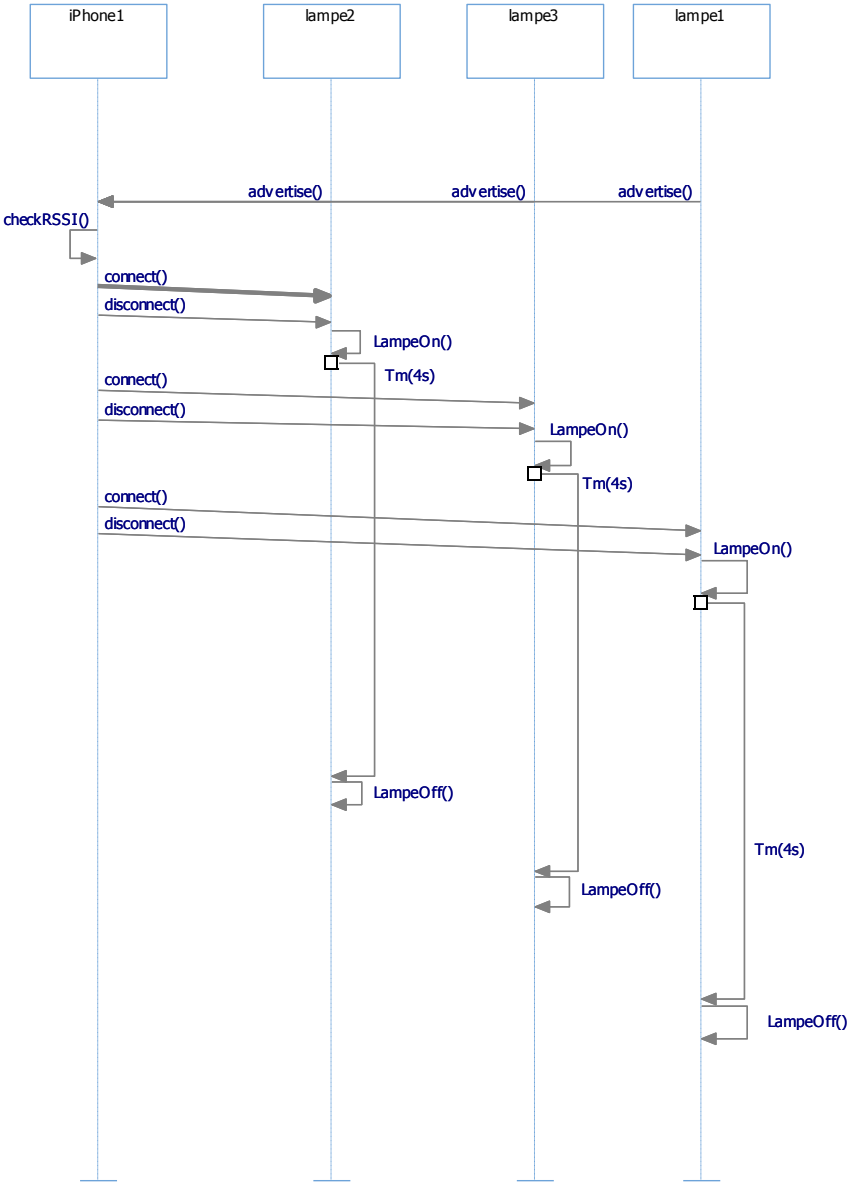


Figure 4 : Lampe adv. : 3 lampes et 1 iPhone : sequence diagram

3.2.3 1 lampe et 3 iPhone

Principe :

Dans ce scénario, il y a donc trois iPhone et une lampe. La lampe va s'annoncer. Puis les iPhone vont se connecter à la lampe les uns après les autres. Le deuxième iPhone devra attendre que le premier iPhone se soit déconnecté pour pouvoir se connecter. Si aucun iPhone ne se connecte durant 4 secondes, la lampe s'éteint.

La figure 5 présente le use case diagramme et la figure 6 le diagramme de séquence.

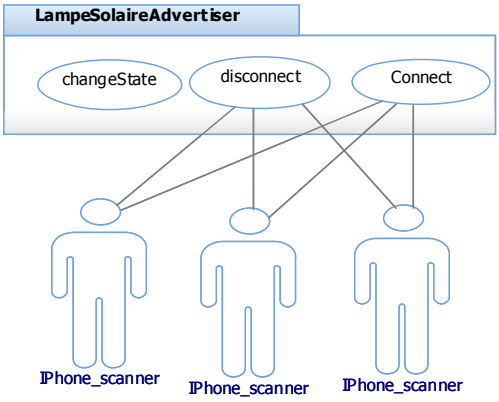


Figure 5 : Lampe adv. : 1 lampe et 3 iPhone : use case diagram

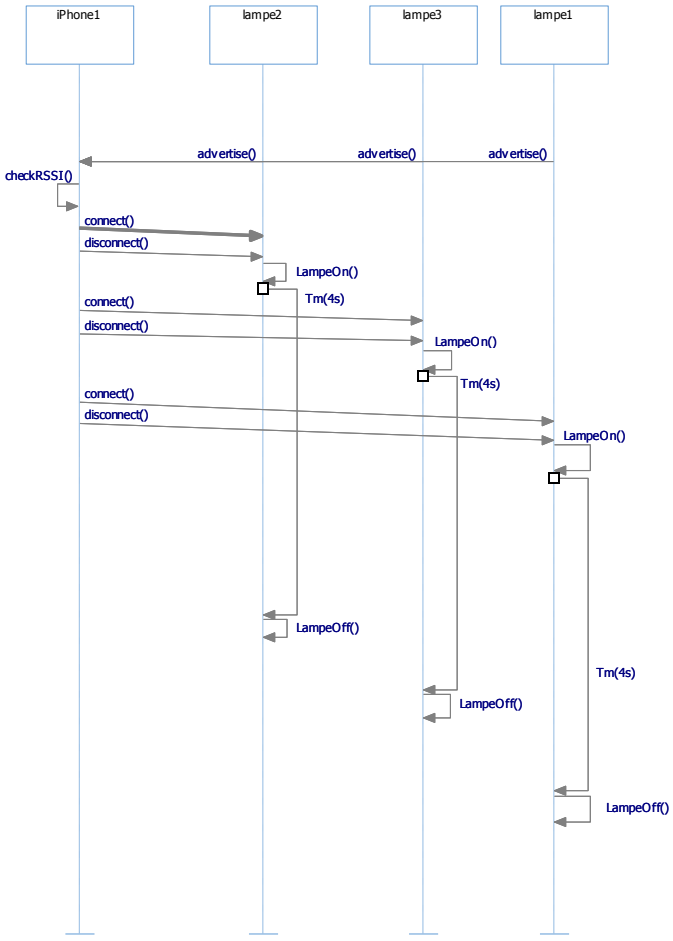


Figure 6 : Lampe adv. : 1 lampe et 3 iPhone : sequence diagram

3.3 SCENARIO LAMPE SOLAIRE SCANNER

Dans ce chapitre, la lampe est le *scanner* et l'*iPhone* est l'*advertiser.*

3.3.1 1 lampe et 1 iPhone

Principe :

Chaque 500ms, l'iPhone va s'annoncer en envoyant son adresse Bluetooth. La lampe qui se situe à proximité va recevoir ce signal, mesurer le RSSI et définir si elle doit s'allumer ou s'éteindre. Si l'iPhone sort de la zone de détection RSSI, le RSSI mesuré par la lampe sera trop faible et elle s'éteindra. Après un certain temps, sans avoir scanné d'annonce d'un advertiser, la lampe va s'éteindre. La figure 7 présente le use case diagramme et la figure 8 le diagramme de séquence.

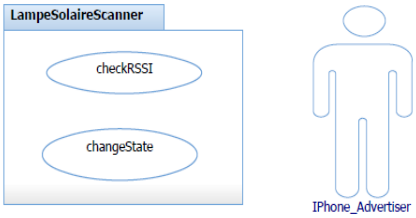


Figure 7 : Lampe scanner : 1 lampe et 1 iPhone : use case diagram

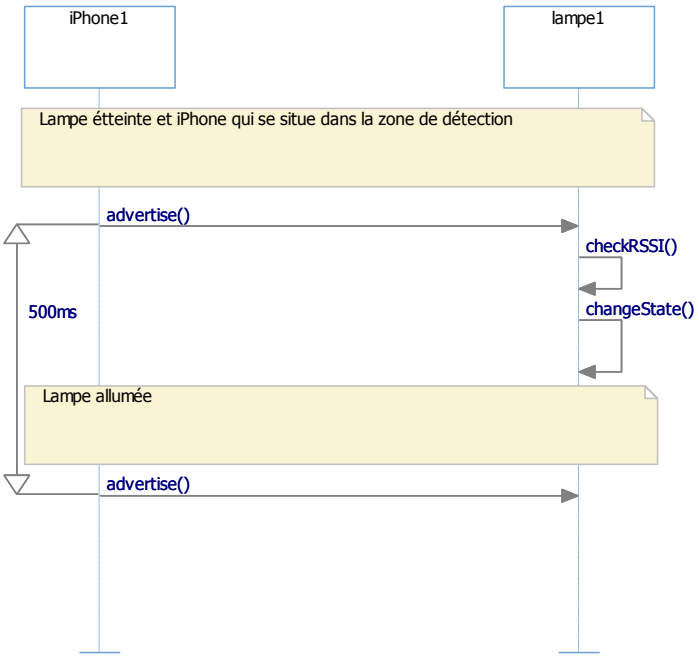


Figure 8 : Lampe scanner : 1 lampe et 1 iPhone : sequence diagram

3.3.2 3 lampes et 1 iPhone

Principe :

Une fois que l'iPhone s'est annoncé, les lampes vont mesurer le RSSI du signal, puis vont s'allumer. La figure 9 présente le use case diagramme et la figure 10 le diagramme de séquence.

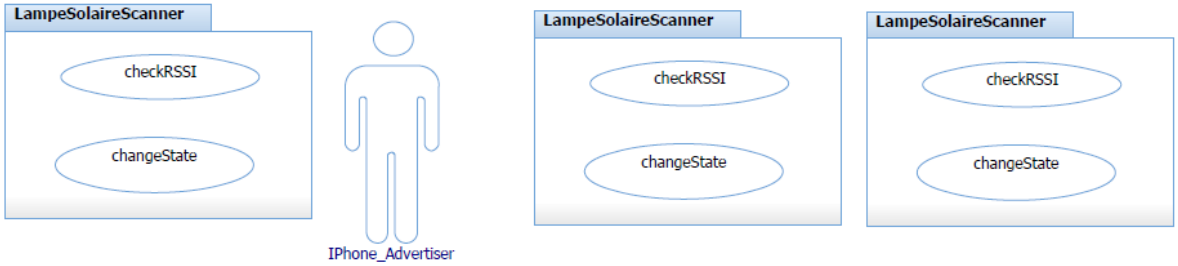


Figure 9 : Lampe scanner : 3 lampes et 1 iPhone : use case diagram

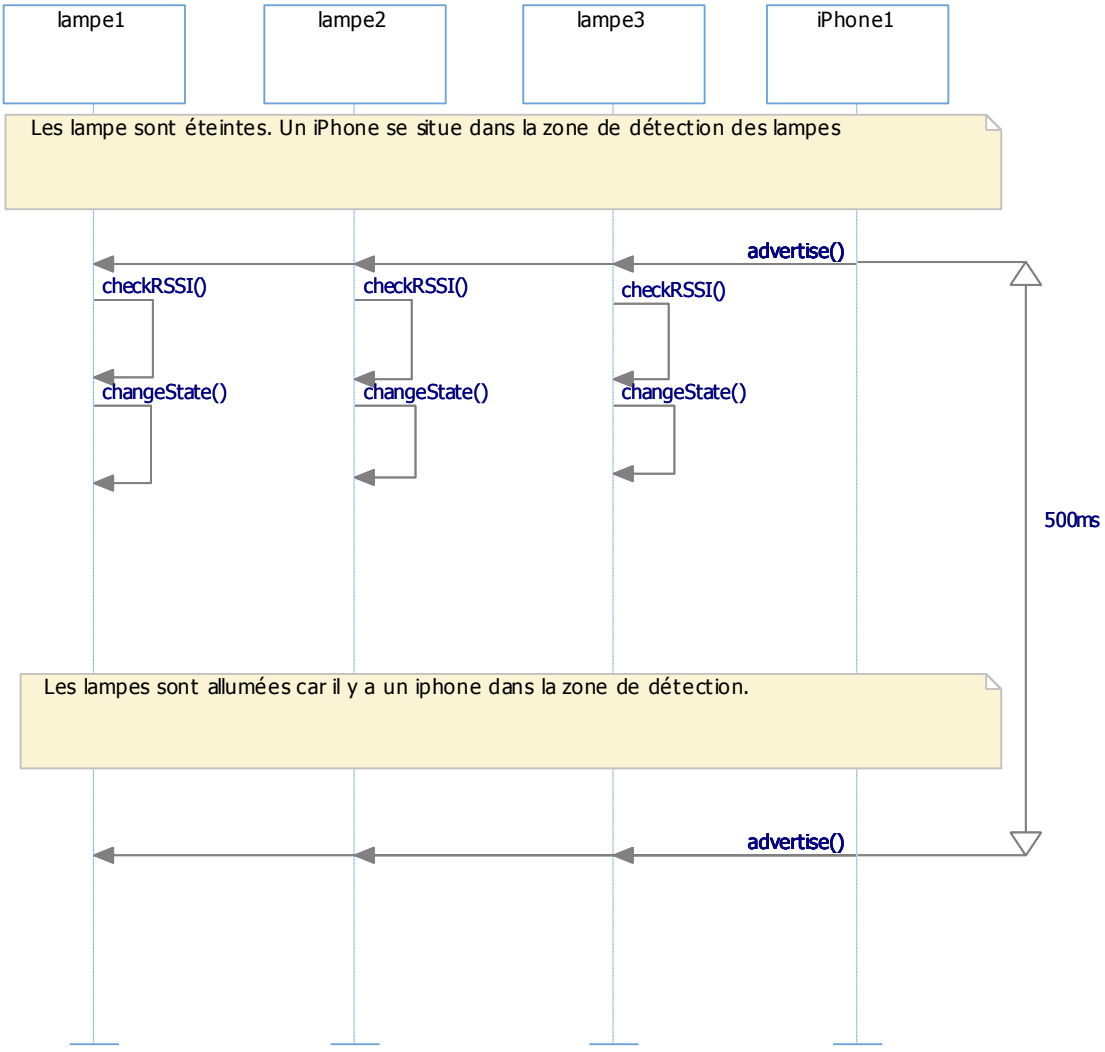


Figure 10 : Lampe scanner : 3 lampes et 1 iPhone : sequence diagram

3.3.3 1 lampe et 3 iPhone

Les iPhone vont s'annoncer, la lampe va mesurer le RSSI du signal puis va s'allumer. Après un certain temps, sans avoir mesuré de RSSI suffisamment puissant, la lampe va s'éteindre. Le cas présenté sur le diagramme de séquence de la figure 13 représente une arrivée désynchronisée des annonces d'iPhone. A noter que si les trois iPhone s'annoncent en même temps, la lampe s'allumera quand même. La figure 11 présente le use case diagramme et la figure 12 le diagramme de séquence.

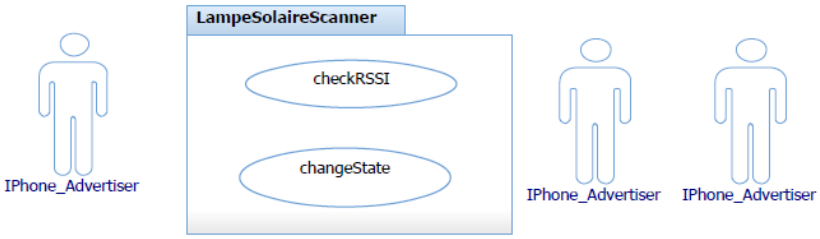


Figure 11 : Lampe scanner : 1 lampe et 3 iPhone : use case diagram

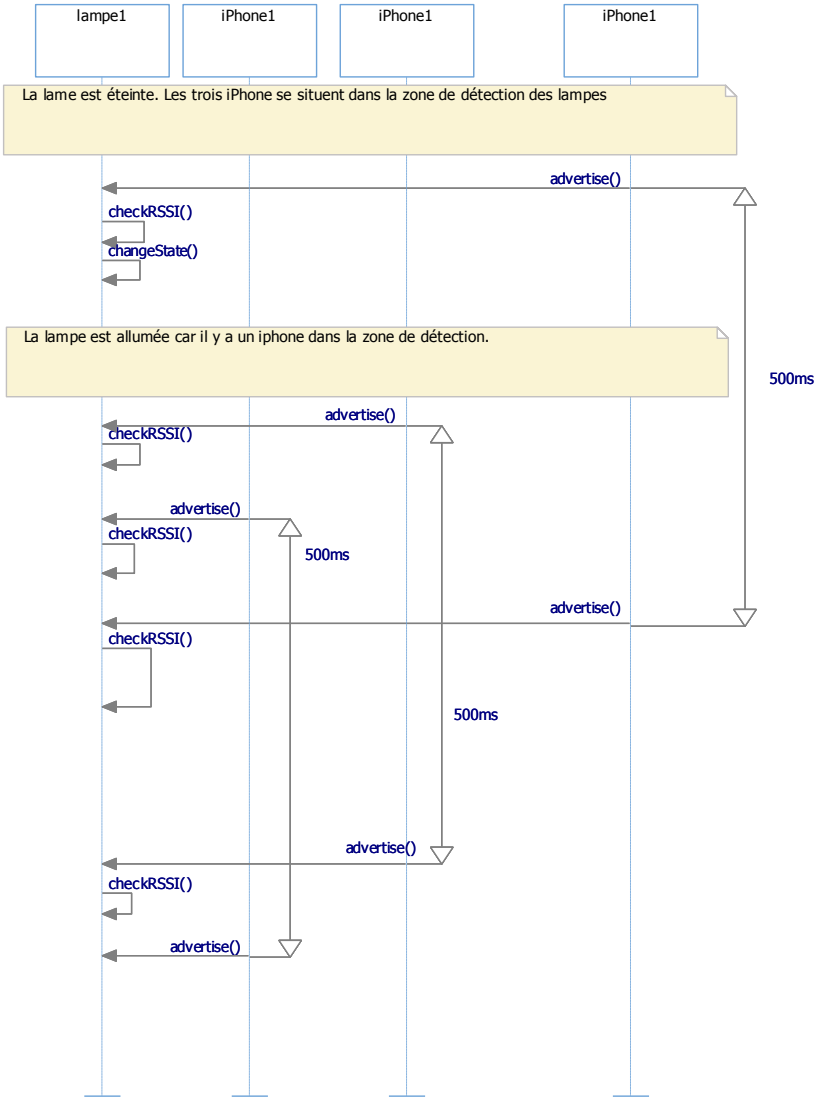


Figure 12 : Lampe scanner : 1 lampe et 3 iPhone : sequence diagram

3.4 Comparaison

Lampe en mode advertiser :

On pourrait également imaginer un chemin le long duquel plusieurs lampes seraient posées. L'application pourrait, par exemple, allumer seulement la lampe la plus proche de l'iPhone. La mesure du RSSI se faisant sur l'iPhone.

Récapitulatif :

Tableau 1 : récapitulatif des avantages et des inconvénients de chaque scénario

	Lampe en advertiser	Lampe en scanner
+	- Possibilité d'allumer seulement la lampe la plus proche de l'iPhone	
o	- Application iPhone plus lourde - Connexion obligatoire	- Application module Bluetooth plus lourde - Pas besoin de connexion
-		

Dans le cadre du fonctionnement du module Bluetooth sur une lampe solaire, il serait intéressant de comparer la consommation des deux modes. Pour effectuer cette mesure, il faudrait programmer les deux rôles et les mesurer. Cette mesure n'a pas été effectuée dans le cadre de ce projet.

La lampe sera en advertiser, car elle offre la possibilité de pouvoir allumer seulement la lampe la plus proche de l'iPhone.

4 BLUETOOTH

4.1 Bluetooth classique vs Bluetooth Low Energy

La technologie Bluetooth Low Energy a été introduite en 2011, grâce à la spécification Bluetooth 4.0. Elle a été développée dans le but de consommer moins d'énergie que le Bluetooth classique.

Un dispositif Bluetooth Low Energy est en mode sommeil et ne se réveille que lorsqu'une connexion est établie. La consommation reste faible même lors d'une connexion, car les temps de connexions réelles sont de seulement quelques millisecondes. La consommation moyenne d'un tel dispositif est généralement de l'ordre de quelques dizaines de micro-ampères.

Toutefois, la technologie Bluetooth classique reste meilleure sur des appareils devant envoyer des flux de données, car il peut avoir des débits de transmission supérieurs au Bluetooth 4.0.

Comme avec le Bluetooth classique, la technologie Bluetooth Low Energy est basée sur un maître relié à un certain nombre d'esclaves. Contrairement au Bluetooth classique qui peut avoir un maximum de 7 slaves, la technologie Bluetooth Low Energy peut avoir un nombre de slaves très important. Cette technologie permet en fait au slave d'avertir le master qu'il a quelque chose à transmettre.

Trois types de classe de module Bluetooth Low Energy existent. Ces classes permettent de déterminer la portée du module Bluetooth. Elles sont les mêmes que pour le Bluetooth classique. Le tableau 2 résume la portée de ces classes. Ce tableau a été pris sur la page Bluetooth de http://fr.wikipedia.org.

Tableau 2 : résumé des différentes classes Bluetooth Low Energy

Classe	Puissance	Portée
1	100 mW (20 dBm)	100 mètres
2	2,5 mW (4 dBm)	10 à 20 mètres
3	1 mW (0 dBm)	quelques mètres

Le Bluetooth Low Energy est déjà présent sur les iPhone depuis la version 4S et sur les iPad depuis la version 2. Google a annoncé que le Bluetooth Low energy sera supporté par la prochaine version d'android. Certains téléphones mobiles android le supportent déjà, tel le HTC ONE.

4.2 Eléments module Bluetooth

Le module Bluetooth utilisé sera le nRF51822. C'est un module Bluetooth qui s'alimente avec une tension pouvant varier de 1.8V à 3.6V. Ce module permet une communication Bluetooth Low Energy. Le nRF51822 est construit autour d'un Cortex M0 ayant 256kB de flash et 16KB de RAM. Il coûte 5 francs/pièce chez Mauser.

Ce module a été choisi, car il est utilisé à l'HES-SO Valais et qu'il a toutes les caractéristiques nécessaires au fonctionnement de ce projet. En effet, il consomme peu de courant et il est compact. Il est sorti en novembre 2012. Ce projet de diplôme permettra donc de tester les possibilités de ce composant.

Il se programme en C. Nordic Semiconductor fournit les librairies, des exemples de service Bluetooth ainsi qu'un système d'exploitation.

Pour développer des software sur le nRF51822, Nordic Semiconductor fournit deux types de kit : le kit de développement et le kit d'évaluation. Le kit de développement dispose de boutons, de différents connecteurs et de différents types d'affichage tandis que le kit d'évaluation dispose de deux boutons, de deux LEDs et de barrettes qui permettent d'accéder à toutes les entrées/sorties du nRF51822.

Nordic Semiconductor fournit également un dongle USB qui est un serveur Bluetooth Low Energy. Ce dernier permet de tester les applications développées sur les différents kits. Le programme Master Control Panel permet de piloter le dongle depuis l'ordinateur.

Le software a tout d'abord été développé sur le kit de développement, puis adapté au kit d'évaluation du nRF51822 et à la board nRF51822 développée lors de ce projet. La figure 13 montre deux kits nordic ainsi que le dongle BLE.

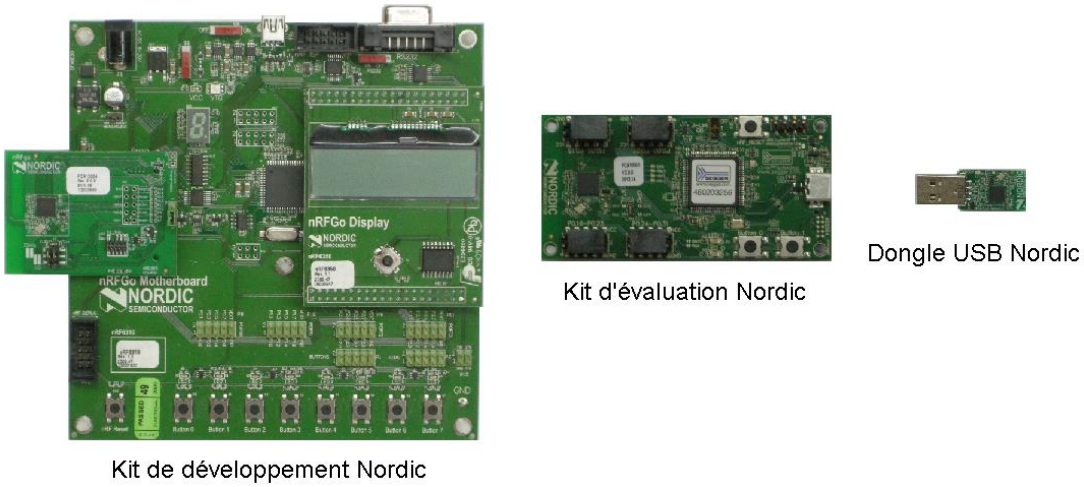


Figure 13 : kits et dongle Nordic

Nordic semiconductor fournit des guides permettant la programmation de codes exemples. Les guides sont téléchargeables sous le site internet http://nordicsemi.com.

4.3 Programmation

Dans cette partie, la programmation de la variante de la lampe solaire utilisée pour expliquer le fonctionnement global est le même pour les deux autres variantes.

4.3.1 Librairie fournie par Nordic Semiconductor

Nordic semiconductor fournit les librairies qui vont s'occuper de la gestion des connections du Bluetooth Low Energy. Le pack de librairie se nomme « softDevice ». La figure 14 montre l'architecture de ces librairies. Cette image est tirée du datasheet du nRF51822.

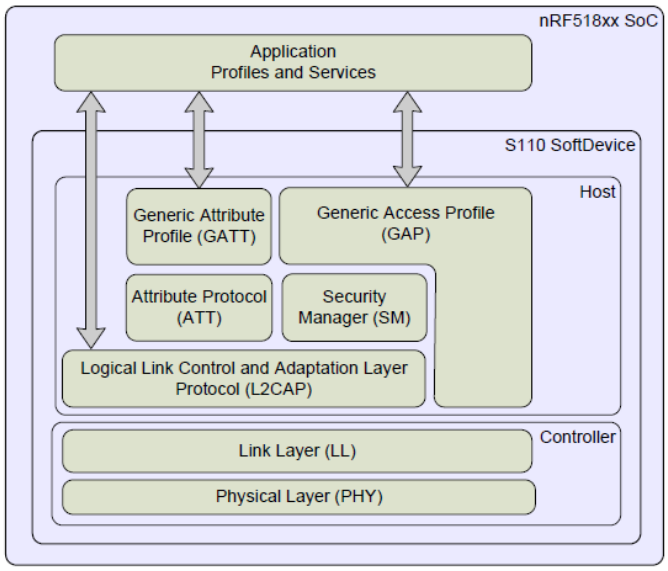


Figure 14 : librairie fournie par Nordic : SoftDevice

L'application développée dans le cadre de ce projet interagira seulement avec le GATT, generic Attribute Profile, et GAP, generic Access Profile. Le GAP définit les procédures génériques de recherche d'appareils, de connexion et de sécurité. Le GATT définit les services dont disposent l'application.

Dans ce chapitre, uniquement les tâches implémentées dans le cadre de ce travail de diplôme vont être décrites.

La programmation a été développée sur le programme Keil. Les drivers et les différents programmes qui permettent la programmation du nRF51822 sont tous fournis par Nordic Semiconductor à l'achat de l'un des deux kits de développement.

La programmation du nRF51822 est divisée en cinq tâches différentes : le mode automatique, le mode manuel, la gestion de l'avertiser, la mesure de la tension de la batterie et la mesure de la tension du panneau solaire.

Le mode automatique permet de piloter une sortie selon le principe étudié dans le chapitre 3 des scénarios de la lampe solaire en advertiser. Le mode manuel permet de piloter les sorties du nRF51822 en envoyant des messages Bluetooth Low Energy depuis l'iPhone. La gestion de l'avertiser permet d'envoyer des burst d'advertise plutôt que d'en envoyer un régulièrement. Il a été jugé intéressant de mesurer la tension des accumulateurs et des panneaux solaires. Cela permet de se faire une idée du chargement de l'accumulateur et de la luminosité sur le panneau solaire.

Des services Bluetooth ont dû être implémentés pour le mode manuel et les deux mesures. Dans ce chapitre, l'implémentation de ces différentes tâches est décrite. Le code complet se trouve dans les annexes de G à L.

4.4 Programmation du nRF51822

Les tests de ce chapitre ont été effectués avec le dongle USB fourni par Nordic, un iPhone 5 et un iPad 2. Le programme master Control pannel de Nordic permet de faire les tests avec le dongle USB, tandis que l'application LightBlue permet de faire les tests avec l'iPhone 5 et iPad 2.

4.4.1 Dispatch Event

Chaque fois que le softDevice reçoit des trames Bluetooth, il va les traiter, puis appelera la méthode ble_evt_dispatch. Cette méthode permet de faire suivre le message reçu aux différents services implémentés et à l'application. La figure 15 explique le fonctionnement de cette méthode.

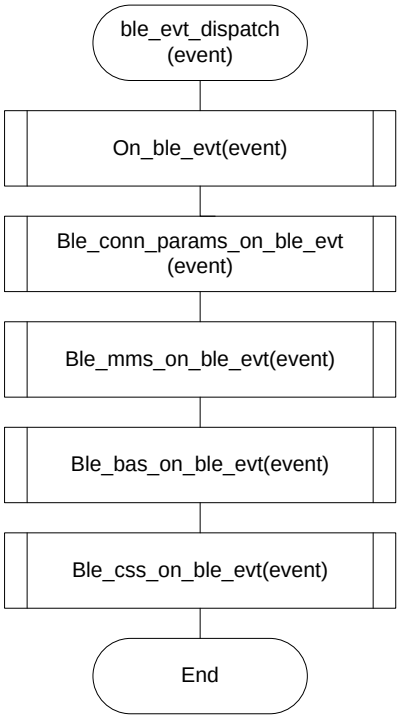


Figure 15 : diagramme de flux de la méthode blue_evt_dispatch

La méthode On_ble_evt est le gestionnaire d'évènement Bluetooth de l'application. Dans le cadre de ce projet, elle s'occupe de gérer le mode automatique.

La méthode Ble_conn_params_on_ble_evt s'occupe de gérer les paramètres de connexion. Elle a été reprise du template que Nordic Semiconductor conseille d'utiliser pour créer un nouveau programme.

Les trois autres méthodes sont les gestionnaires d'évènement Bluetooth des trois services implémentés durant ce projet. Le service MMS, Manual Mode Service, gère le mode manuel, le service BAS, Battery Service, gère la mesure de la batterie et le service CSS, Cell Solar Service, gère la mesure de la cellule solaire.

4.4.2 Mode automatique

Principe

Le mode automatique permet de piloter une sortie comme expliqué dans les scénarios du chapitre 3 de la lampe solaire en advertiser. La figure 16 explique le fonctionnement du mode automatique.

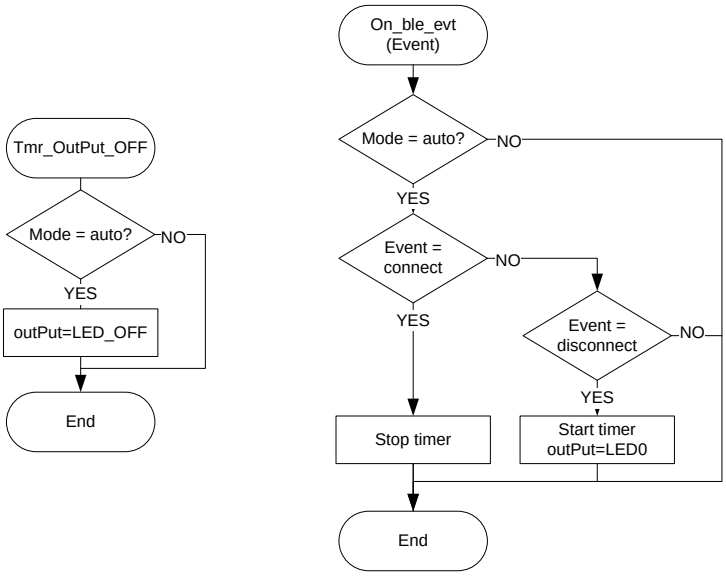


Figure 16 : diagramme de flux du mode automatique

Chaque fois qu'un événement Bluetooth apparaît, la méthode On_ble_evt est appelée par le dispatch_ble_evt expliqué au chapitre 4.4.1. A la déconnexion d'un serveur, un timer va être lancé et la led va être activée. La led s'éteint si, après un certain temps, aucun serveur ne s'est connecté. Ce temps est défini grâce au timer « Tmr_OutPut_OFF». Ce timer est stoppé à chaque fois qu'un serveur se connecte.

Au moment de la connexion d'un serveur, il n'y a pas de possibilité de savoir ce que veut faire l'iPhone. Il se peut qu'il ne veuille pas allumer la LED, mais juste récupérer le niveau de la batterie. Pour parer se problème, l'allumage de la LED se fait à la déconnection du serveur.

Paramètre du timer

```
1   #define OUTPUT_OFF_TIMER_INTERVAL
                            APP_TIMER_TICKS(4000, APP_TIMER_PRESCALER)
```

Ce paramètre détermine la durée en millisecondes du timer « Tmr_OutPut_OFF ». Dans le cas ci-dessus, après 4 secondes sans connexion d'un iPhone, la led va s'éteindre.

Tests

La LED s'allume à la déconnexion d'un serveur. Tant que quelqu'un se connecte durant un délai de 4 secondes, la LED reste allumée. Si personne ne se connecte dans ce délai, elle s'éteint. Un deuxième test a été effectué avec trois appareils différents qui se connectent à tour de rôle (un iPad, un iPhone et un dongle USB de test fourni par Nordic). La LED reste allumée. Cette partie fonctionne donc correctement.

4.4.3 Mode manuel

Service

Un service a du être implémenté afin de pouvoir piloter les sorties du nRF51822 grâce à un message Bluetooth Low Energy. Ce service se nomme MMS. Il est inspiré du service IAS (Immediate Alert Service) fourni dans les exemples de service Bluetooth de Nordic Semiconductor.

Principe

Le mode manuel permet de piloter les sorties du nRF51822 en envoyant des messages Bluetooth Low Energy depuis l'iPhone. Dans le cadre de ce projet, seulement deux sorties sont pilotées. Elles prennent la valeur du message reçu. La figure 17 explique le principe de fonctionnement du mode manuel.

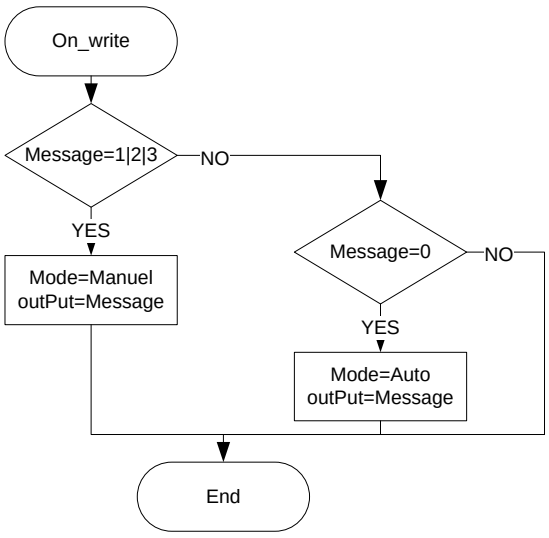


Figure 17 : diagramme de flux de la fonction on_mms

La méthode on_mms est appelée depuis le dispatch d'event expliqué au chapitre 4.4.1. Le mode manuel est enclenché dès qu'un message Bluetooth destiné au service MMS est reçu. Le système repasse en mode automatique quand un message ayant la valeur 0x00 est reçu.

Tests

Afin de tester ce service, l'application LightBlue a été utilisée. Le service du mode manuel n'étant pas connu de l'iPhone, le service Alert Notification Service a été choisi, car il est le service prédéfini se rapprochant le plus dans son fonctionnement. La figure 18 montre l'application LightBlue. Une fois le IAS sélectionné, il suffit d'y insérer la valeur des sorties désirées.

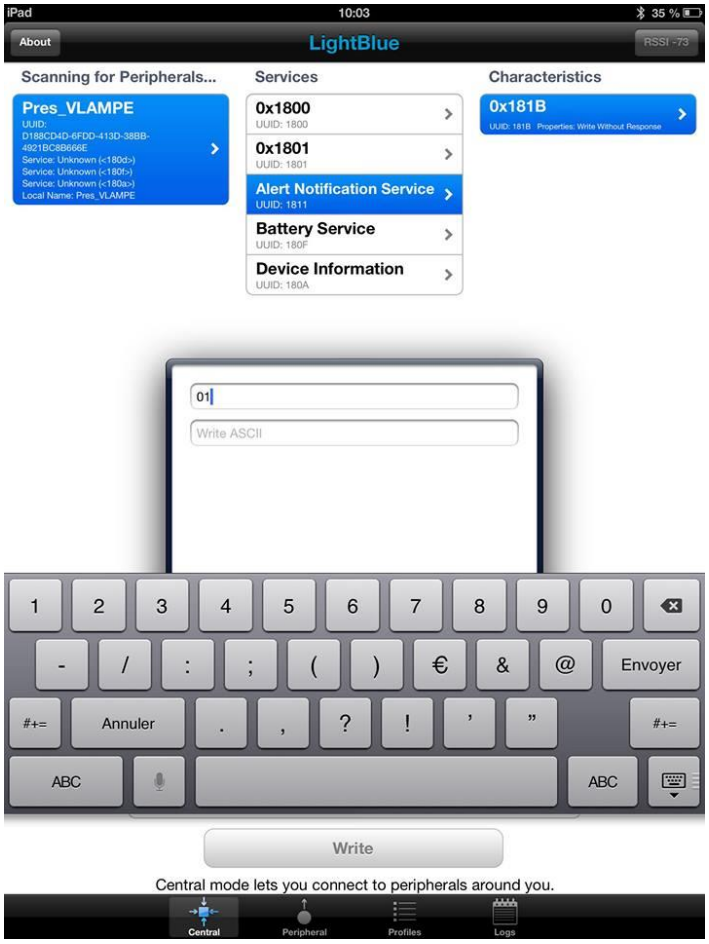


Figure 18 : Test avec lightBlue du fonctionnement du mode automatique

Les LED s'allument selon le message envoyé :

- 0x00= aucune
- 0x01= LED0
- 0x02= LED1
- 0x03= LED0 & LED1

Si le serveur se déconnecte, les LEDs allumées restent allumées. Pour les éteindre, il faut qu'un serveur se connecte et envoie le message 0x00.

Une fois un serveur connecté, aucun autre serveur ne peut se connecter, car le module Bluetooth n'est plus visible pour les autres serveurs. Une fois déconnecté, n'importe quel serveur Bluetooth peut venir se connecter afin de modifier les sorties. Ce test a été effectué avec un iPhone, un iPad et le dongle USB que Nordic fournit pour les tests. Lors de la réception d'un message différent de 0x00, le mode automatique est désactivé. Lorsque le message 0x00 est reçu, le mode automatique est relancé. Cette partie fonctionne correctement.

4.4.4 Mesures de la tension de la batterie et du panneau solaire

Service

Le service pour la mesure de la batterie est fourni dans les exemples Nordic. Un service a été implémenté pour la mesure du panneau solaire. Ce service s'inspire du service de la batterie. Le service pour la mesure de la batterie se nomme « BAS » (batterie service) et celui pour la mesure de la cellule solaire se nomme « CSS » (cellule solar service). Ces services ne seront pas utilisés dans la variante prise murale. Les deux prototypes contenant ces services sont donc la variante adaptable à une lampe solaire et la variante de la lampe solaire.

Principe

A chaque connexion, une mesure de la tension de la cellule solaire et de la tension de la batterie est effectuée. Une fois connecté, un timer est lancé et si le serveur reste connecté suffisamment longtemps, les deux mesures sont reproduites. Il y a donc deux possibilités de lancer une mesure. La figure 19 explique le fonctionnement de ces mesures.

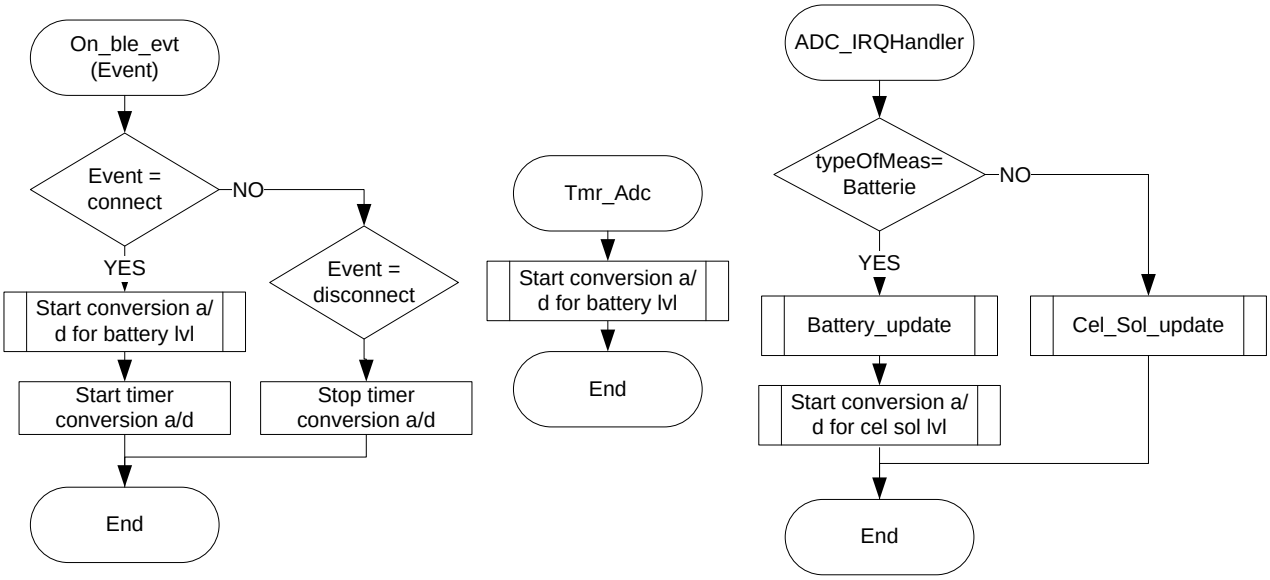


Figure 19 : diagramme de flux des conversions AD

Une seule conversion AD peut être effectuée en même temps. La conversion de la batterie est lancée en premier. Une fois terminée, l'interruption « ADC_IRQHandler » est appelée. Cette interruption va mettre à jour la valeur de la tension de la batterie dans le service « BAS ». Ce service va ensuite la placer dans le gatts, afin que le softDevice puisse l'envoyer au serveur si ce dernier désire la connaître.

L'interruption « ADC_IRQHandler » va ensuite lancer la conversion AD de la mesure de la cellule solaire. A la fin de la conversion, l'interruption ADC_IRQHandler va réapparaître et va mettre à jour le service « CSS » qui fonctionne de la même façon que le service « BAS ».

Paramètre du timer

```
1   #define ADCCONVERT_LEVEL_MEAS_INTERVAL APP_TIMER_TICKS
                                            (30000, APP_TIMER_PRESCALER)
```

Cette constante permet de définir le temps entre deux prises de mesure si un appareil reste connecté. Dans le cas ci-dessus, elle se situe à 30 secondes.

Paramètre du convertisseur AD

Les conversions AD peuvent se faire sur 8, 9 ou 10 bits. La valeur retournée par le convertisseur AD varie de 0 à $2^n - 2$ (n étant le nombre de bits sur lesquels s'effectue la conversion). $2^n - 1$ correspond à une erreur de conversion. $2^n - 2$ correspond à la tension de référence. Il y a quatre possibilités de tensions de référence :

- 1.2V créé en interne
- La tension d'alimentation du nRF51822
- La tension sur l'entrée p0.00
- La tension sur l'entrée p0.10

Dans le cadre de ce projet, la tension de référence sera 1.2V.

Le calcul ci-dessous permet de trouver la résolution de la mesure avec un convertisseur 8 bits :

$$\text{Résolution}\left[\text{V}\right] = \frac{1*U\max}{2^n - 2} = \frac{1*1.2}{2^8 - 2} = 4.72mV$$

Une conversion 8 bits a été jugée suffisamment précise pour permettre de se faire une idée de la tension se trouvant aux bornes de l'accumulateur et du panneau solaire.

Afin de faciliter la lecture de la tension sur l'iPhone, une transformation a été effectuée sur la valeur envoyée à ce dernier. Le but étant de pouvoir facilement convertir la valeur affichée par l'application LightBlue de l'iPhone sans utiliser de calculatrice.

Pour effectuer cette conversion, des constantes sont utilisées. Ces constantes varient en fonction de la variante sur laquelle la mesure est effectuée.

Traitement des mesures de la variante bq25504

```
1   #define     FACTOR_CELL_SOL_LVL_IN_MILLIVOLT  1.2/254*100
2   #define     FACTOR_BATTERY_LVL_IN_MILLIVOLT   1.2/254*100/0.25/2
```

Le panneau solaire variant de 0V à 0.5V, il n'est pas nécessaire de passer par un pont diviseur pour le mesurer. La conversion AD étant linéaire, une simple multiplication permet d'obtenir la tension réelle du panneau solaire depuis la valeur du convertisseur AD. Le facteur a été défini de la manière suivante:

$$FACTOR\_CELL\_SOL = \frac{U\max}{Valeur\max} * 100 = \frac{1.2}{254} * 100 = 0.471$$

Une multiplication par 100 a été faite afin d'avoir des nombres naturels, car le stockage se fait en uint8 et non pas en float. Ce facteur permet d'obtenir la mesure de la conversion AD en mV/10. Exemple : lorsqu'il y a 0.260V aux bornes du panneau, l'iPhone indiquera 26.

Le facteur de multiplication de la mesure de la cellule solaire est réutilisé pour définir celui de la mesure de la batterie.

Comme la tension de la batterie est divisée par un pont diviseur avant d'arriver sur le convertisseur AD, le facteur est divisé par le gain du pont diviseur. Dans le cas de la variante lampe solaire, le facteur est divisé par 0.25. Le tout est divisé par deux, ceci afin que la valeur maximale, 4.2V, puisse être stockée dans un Uint8. Lorsqu'il y a 4.2V aux bornes de l'accumulateur, l'application Light Blue de l'iPhone indiquera 210.

Traitement des mesures de la version adaptée à une lampe solaire

Dans cette version, la tension des accumulateurs varie dans la même gamme de tension que la tension du panneau solaire. Le pont diviseur de l'accumulateur et du panneau solaire est le même. Le facteur de multiplication sera donc le même pour les deux mesures.

```
1   #define     FACTOR_CELL_SOL_LVL_IN_MILLIVOLT        1.2/254*100*0.4
2   #define     FACTOR_BATTERY_LVL_IN_MILLIVOLT         1.2/254*100*0.4
```

Les caractéristiques de la conversion AD sont les mêmes que celle du traitement de la mesure de la tension de l'accumulateur de la variante bq25504. Le facteur de conversion est donc calculé grâce à la formule suivante :

$$FACTOR\_ADC = \frac{U\max}{Valeur\max} * 100 = \frac{1.2}{254} * 100 = 0.471$$

Comme la tension de batterie est divisée par un pont diviseur avant d'arriver sur le convertisseur AD, le facteur est divisé par le gain du pont diviseur. Dans le cas de la variante adaptée à une lampe solaire, le facteur est divisé par 0.4. La tension de l'accumulateur et celle du panneau solaire ne dépassent pas 2.54V. Cette valeur peut être stockée dans un uint8. Lorsqu'il y a 2V sur le panneau, l'application Light Blue de l'iPhone indiquera 200.

Tests

Sur LigthBlue, les services « Battery Service » et « Device Information » sont les services des deux mesures. Une fois ces services sélectionnés, il suffit de lire la valeur décimale. Ces valeurs sont encadrées en rouge sur la figure 20. Dans le cas ci-dessous, il y a 2.41V aux bornes des accumulateurs et 680mV aux bornes du panneau solaire. Ces mesures ont été prises sur la variante adaptée à une lampe solaire.

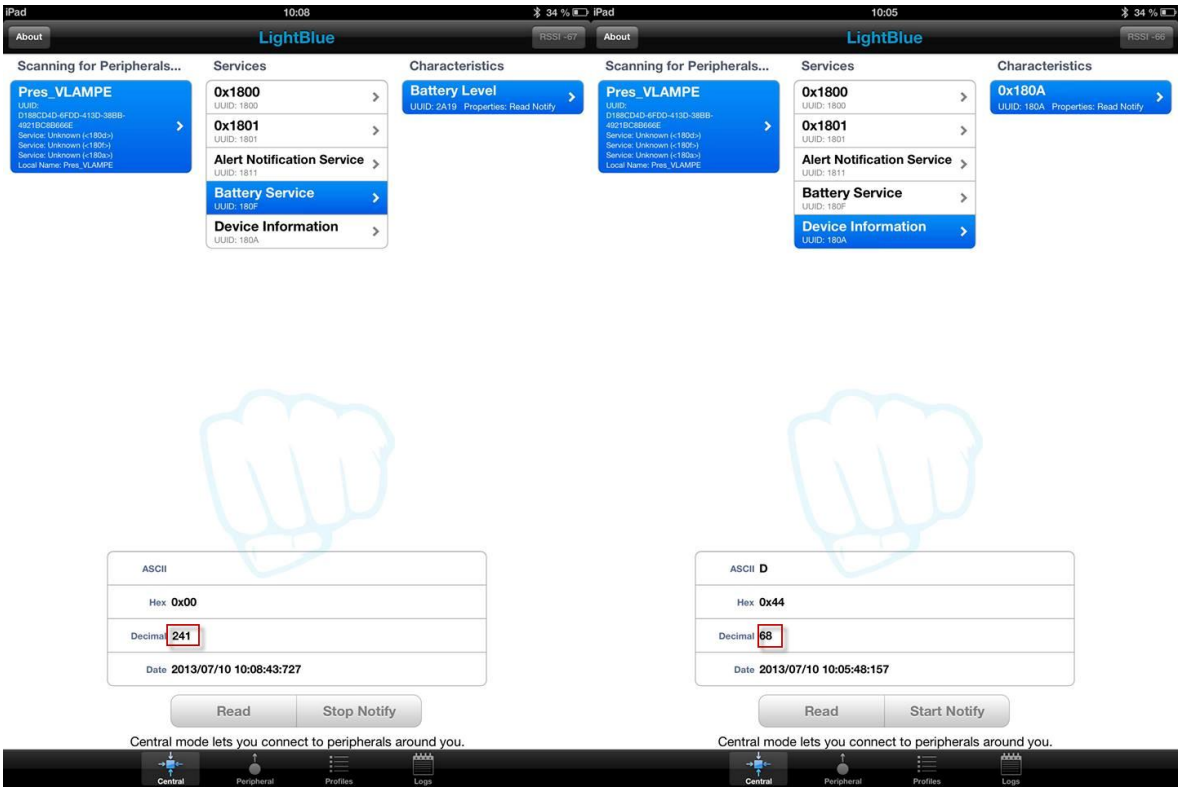


Figure 20 : Test avec LightBlue du fonctionnement des mesures

Pour tester si les deux services, CSS et BAS, fonctionnent correctement, la tension de la batterie et du panneau solaire est simulée par une alimentation de laboratoire. Cette tension sera mesurée par le convertisseur AD, modifiée par les facteurs calculés plus haut, puis envoyée sur l'iPhone.

Les valeurs du tableau 3 sont les valeurs lues sur l'application LightBlue.

Tableau 3 : Valeurs des mesures reçues par l'iPhone

	Variante adaptée à une Lampe solaire		Variante bq25504	
Uin [V]	CSS [-]	BAT [-]	CSS [-]	BAT [-]
0	2	2	0	17
0.5	50	50	49	27
1	98	102	99	52
1.5	149	151		76
2	200	200		99
2.5	246	246		123
3				147
3.5				170
4				192
4.2				200
err moy [-]:	2	2	0.67	10.2
err moy [V]:	20mV	20mV	6.7mV	102mV

Pour la variante lampe solaire, la même erreur moyenne est obtenue pour les deux mesures. Cette erreur est de 20mV. Cette erreur représente une erreur de 0.8% si la tension mesurée est de 2.5V; ce qui suffit amplement pour se faire une idée de la tension aux bornes du panneau solaire et de l'accumulateur.

Pour la variante bq25504, le panneau solaire est directement relié sur le nRF51822. L'erreur moyenne de 6.7mV est due à des arrondis. Pour la batterie, l'erreur est plus grande. Le fait d'avoir un pont diviseur ayant une résistance globale de 80kΩ en est la cause. Pour plus de détails, il faut se référer au chapitre 6.2.2.

La figure 21 représente l'erreur des différentes conversions AD :

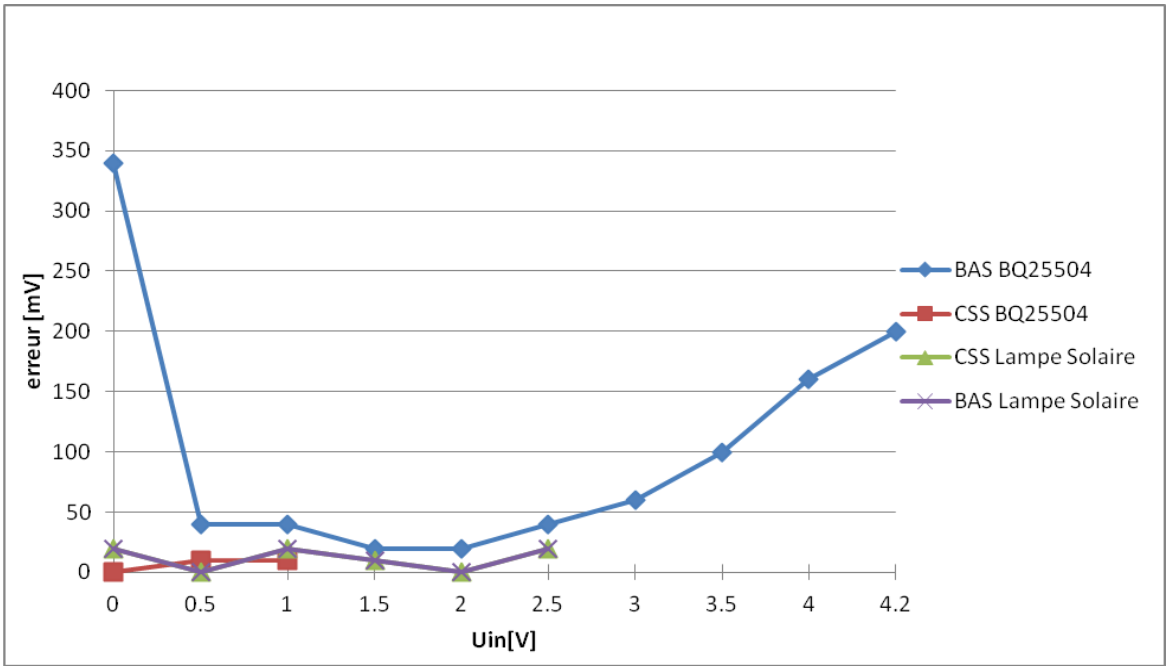


Figure 21 : erreur de mesures des services MMS et BAS

Ce graphique permet de voir que l'erreur du service CSS de la variante lampe solaire et des services CSS et BAS de la variante adaptable à une lampe solaire est stable. L'erreur pour les deux mesures du bq25504 est identique. Ceci est dû au fait que le même pont diviseur est utilisé.

L'erreur du service BAS de la lampe solaire varie beaucoup. La tension mesurée sur l'accumulateur peut varier de 3.2V à 4.2V. L'erreur dans cette partie n'est pas stable. Elle varie de 70mV à 200mV. L'erreur relative la plus grande est de 4.7% pour une mesure de 4.2V. Afin d'améliorer cette moyenne, il faudrait pouvoir couper le pont diviseur à l'aide d'un JFET lorsqu'aucune conversion AD n'est effectuée.

4.4.5 Gestion de l'advertiser

Principe

Le datasheet du nRF51822 indique que des pics de courant de 10mA apparaissent à chaque fois qu'il advertise. Afin d'économiser de l'énergie, un burst d'advertise est envoyé à intervalles réguliers plutôt que d'envoyer des advertise en permanence.

C'est la sous-couche GAP, Generic Access Profile, du softDevice qui va gérer l'advertiser. Le processeur peut activer ou désactiver l'advertiser ainsi que choisir le temps entre chaque advertise. La figure 22 montre le diagramme de flux de cette partie.

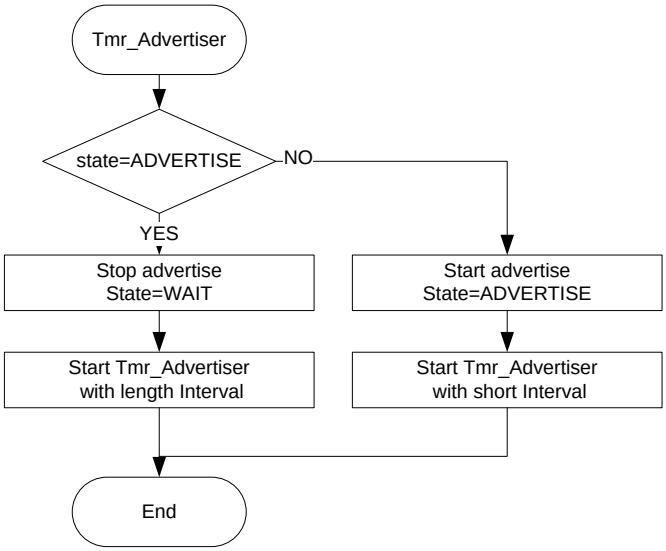


Figure 22 : diagramme de flux du timer gérant l'advertiser

Le principe est d'enclencher l'envoi d'advertise en même temps qu'un timer de l'ordre de plusieurs dizaines de millisecondes. Une fois ce timer arrivé à son terme, le module stoppe l'envoi d'advertise. Un timer de l'ordre de quelques secondes est enclenché. Ce timer permet d'attendre quelques secondes avant de renvoyer un burst.

Pour se faire un seul timer est utilisé, sa durée est modifiée selon le mode dans lequel se trouve la sous-couche GAP.

Paramètre de la gestion du burst

Deux paramètres permettent de régler les deux durées différentes du timer :

```
1   #define ADVERTISER_TIMER_INTERVAL_LENGTH
                          APP_TIMER_TICKS(2000,APP_TIMER_PRESCALER)
2   #define ADVERTISER_TIMER_INTERVAL_SHORT
          APP_TIMER_TICKS(NBR_OF_ADVERTISE_IN_BURST*APP_ADV_INTERVAL*
          625/1000, APP_TIMER_PRESCALER)
```

La figure 23 explique la différence entre ces deux durées:

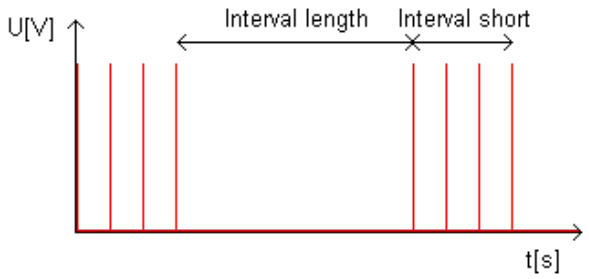


Figure 23 : Fonctionnement des différents timer du mode burst

Sur la figure 23, les pics de tension représentent les advertise. La durée de l'intervalle long est directement définie dans la constante `ADVERTISER_TIMER_INTERVAL_LENGTH`(ligne 1).

Un calcul utilisant d'autres constantes est réalisé afin de simplifier la modification de l'intervalle court. Ce calcul est réalisé grâce aux deux constantes ci-dessous :

```
3   #define NBR_OF_ADVERTISE_IN_BURST      5
4   #define APP_ADV_INTERVAL               32
```

La sous-couche GAP advertise sur une base de temps de 0.625ms. La constante APP_ADV_INTERVAL (ligne 4) définit le nombre de ticks que doit attendre le GAP avant de relancer un advertise. On peut définir le temps entre deux advertise avec l'équation suivante :

INTERVAL_ADV=APP_ADV_INTERVAL*0.625=32*0.625=20ms

Chaque 20ms, le nRF51822 va envoyer un advertise. Cette valeur a été définie après une série de tests. En dessous de 13ms, les différents serveurs, iPhone, iPad et le dongle Nordic, ne détectaient pas le nRF51822. Une marge d'erreur de 7ms a été adoptée.

La constante NBR_OF_ADVERTISE_IN_BURST(ligne 3) définit le nombre d'advertise à envoyer par burst.

La constante ADVERTISER_TIMER_INTERVAL_SHORT (ligne 2) définit la durée pendant laquelle le nRF51822 advertise. Elle est définie comme suit :

ADVERTISER_TIMER_INTVERVAL_SHORT=INTERVAL_ADV*NBR_OF_ADVERTISE_IN_BURST

Dans notre cas, si l'on veut envoyer 5 advertise dans le burst, la durée du timer sera de 100ms.

Test

Ce test a été effectué sur la version lampe solaire. La mesure a été prise sur l'antenne Bluetooth avec un oscilloscope Agilent de type MSO-X 3012A. Le test permet de voir quand le nRF51822 advertise. En revanche, il ne permet pas de voir les trames Bluetooth qui se situent à 2.4GHz et qui se trouvent en dehors de la plage de fréquences de l'oscilloscope.

Le test a été réalisé avec 5 advertise par burst et un espacement de 2 secondes entre deux burst.

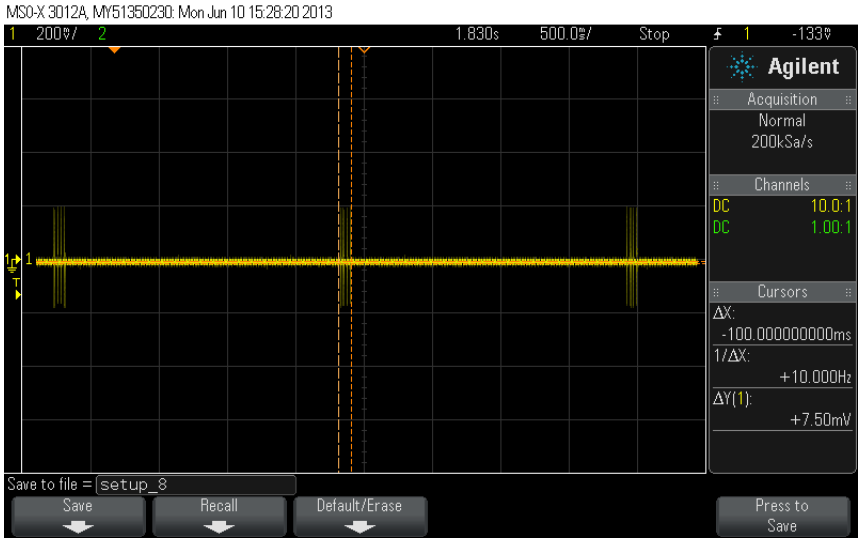


Figure 24 : Mesure de l'intervalle court avec 5 advertise par burst

Sur la figure 24, on peut voir que la durée du timer court est de 100ms. Ce qui correspond au résultat attendu : 5*0.625*32=100ms. 5 pics sont présents.

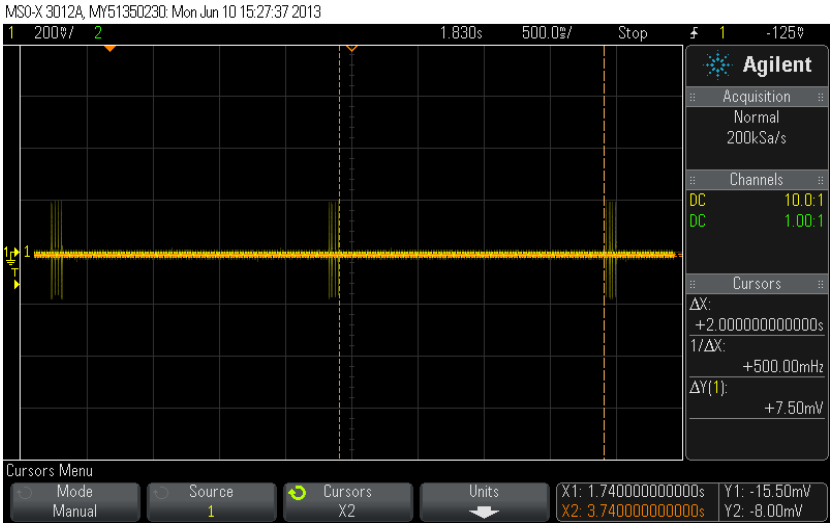


Figure 25 : Mesure de l'intervalle long avec 2 secondes

Sur la figure 25, on peut voir que la durée du timer long est de 2 secondes. Cette valeur correspond au résultat attendu.

5 BOARD NRF51822

5.1 Principe

Une board contenant uniquement le nRF51822 et les composants essentiels à son bon fonctionnement a été développée. Cette board se décompose en différentes parties. Le schéma bloc de cette board se trouve sur la figure 26:

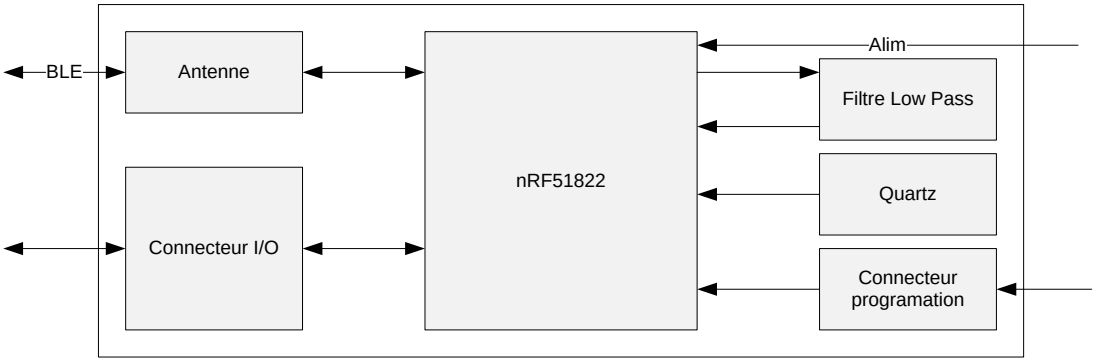


Figure 26 : Schéma de principe de la board du nRF51822

Les différentes parties de ce schéma bloc sont décrits dans ce chapitre.

Le schéma électrique compet de la board se trouve en annexe B.

5.2 Développement

Nordic Semiconductor fournit le schéma éléctrique et le PCB de tous ses kits de développement. Le schéma électrique a été inspiré du datasheet fourni avec le kit de développement du nRF51822. C'est le seul datasheet des kits du nRF51822 n'ayant pas l'indication « NOT TO BE USED AS REFERENCE LAYOUT ».

5.2.1 nRF51822

Le nRF51822 existe en boitier WLCSP et en boitier QFN48. Le défaut du boitier en WLCSP réside dans le fait que les pins se trouvent dessous le composant et nécessitent un PCB professionnel de très bonne qualité, car l'espacement entre deux balls est de 0.4mm. De ce fait, lors de ce projet, le boitier QFN48 sera utilisé.

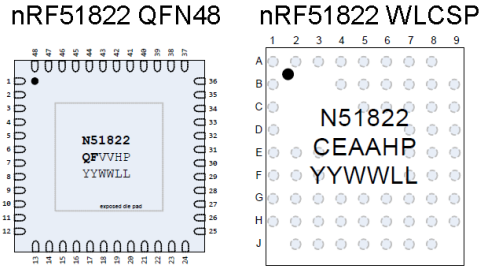


Figure 27 : boitier du nRF51822

5.2.2 Antenne

L'antenne est créée avec une piste sur le PCB. Ses dimensions ont été récupérées grâce au fichier Gerber du kit de développement.

Les valeurs des composants permettant l'adaptation de l'antenne ont été récupérées sur le datasheet du kit de développement du nRF51822. STMicroelectronics a sorti un Balun qui permet d'adapter l'antenne sans utiliser de résistances, de bobines et de condensateurs. L'avantage du balun est qu'il est petit et qu'il adapte l'antenne de façon plus précise qu'avec l'utilisation de composants. Le problème du balun étant qu'il avait un délai de livraison de 6 semaines. De ce fait, il n'a pas été utilisé dans le cadre de ce projet. La référence du balun est le NRF01D3.

A noter qu'une attention particulière à l'alignement des composants de l'adaptation d'impédance a été faite lors du routage. Toutefois les composants sont plus éloignés que sur le kit. Des composants en boitier 0603 sont utilisés sur la board alors que le kit utilise des composants en 0205 ce qui peut influencer l'adaptation de l'antenne.

5.2.3 Quartz

Un quartz de 16MHz est obligatoire afin de faire fonctionner le nRF51822. A noter que le quartz choisi nécessite deux condensateurs de 12pF pour entrer en résonnance.

Un deuxième quartz de 32.768kHz peut être ajouté. Ce deuxième quartz n'est pas obligatoire, car un module interne du nRF51822 peut le générer. La génération en interne du quartz permet de diminuer les coûts de production, mais augmente la consommation du nRF51822. Dans le cadre de ce projet, la deuxième fréquence d'oscillation de 32.768kHz sera générée en interne.

5.2.4 Filtre low Pass

Le nRF51822 contient un convertisseur DC/DC qui transforme la tension d'alimentation en une tension plus faible. Il permet de diminuer la consommation du nRF51822.

Pour utiliser ce convertisseur DC/DC, il faut mettre un filtre passe-bas entre la pin DCC et les pins AVDD du nRF51822. La figure 28 montre le chemin du courant qui va alimenter le nRF51822.

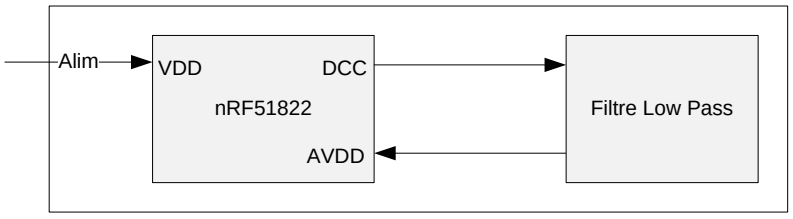


Figure 28 : Principe du filtre low Pass

D'après le datasheet, ce filtre passe-bas permet de diminuer la consommation d'approximativement 30%.

5.2.5 Connecteur de programmation

Le module de programmation du kit de développement est utilisé pour programmer le nRF51822. Les différentes connexions du connecteur sont inspirées de celles du datasheet de ce kit.

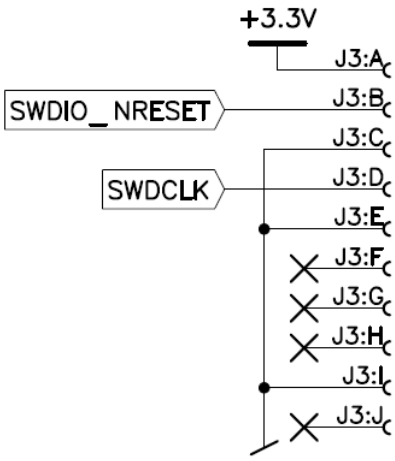


Figure 29 : Connecteur de programmation

Une modification a été faite sur le schéma fourni. La pin A de ce connecteur a été reliée au VCC, car les logiciels nRFStudio et Keil prennent une mesure de la tension se situant sur la pin 1 du connecteur avant d'effectuer la programmation.

5.2.6 Connecteur I/O

Afin d'accéder aux différentes entrées et sorties du nRF51822, deux connecteurs sont présents sur la board du nRF51822. Toutes les entrées et sorties du nRF51822 ne se trouvent pas sur les connecteurs. L'alimentation de la board nRF51822 se fera à travers ces deux connecteurs. Le tableau 4 résume le pinning des deux connecteurs :

Tableau 4 : pinning des connecteurs j1 et j2

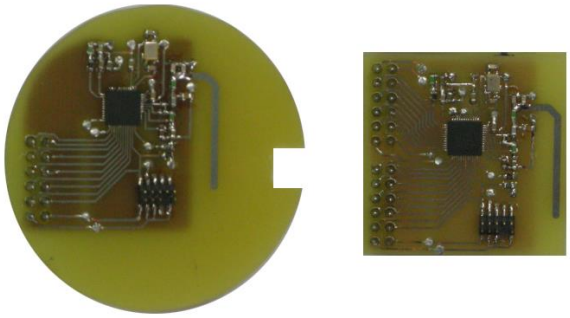
	Connecteur J1	
Pin con.	**Pin nRF51822**	**Description**
1	P0.26	General purpose I/O pin ADC input 0 Connection for 32.768 kHz crystal
2	P0.27	General purpose I/O pin ADC input 1 Connection for 32.768 kHz crystal or external 32.768 kHz clock reference
3	P0.00	General purpose I/O pin ADC Reference voltage
4	P0.01	General purpose I/O pin ADC input 2
5	P0.02	General purpose I/O pin ADC input 3
6	P0.03	General purpose I/O pin ADC input 4
7	P0.04	General purpose I/O pin ADC input 5
8	P0.05	General purpose I/O pin ADC input 6
9	P0.06	General purpose I/O pin ADC input 7 ADC Reference voltage
10	P0.07	General purpose I/O pin
11	GND	
12	VCC	
	Connecteur J2	
Pin con.	**Pin nRF51822**	**Description**
1	P0.08	General purpose I/O pin
2	P0.09	General purpose I/O pin
3	P0.10	General purpose I/O pin
4	P0.11	General purpose I/O pin
5	P0.12	General purpose I/O pin
6	P0.13	General purpose I/O pin
7	P0.14	General purpose I/O pin
8	P0.15	General purpose I/O pin
9	P0.16	General purpose I/O pin
10	SWDIO/nRESET	System reset (active low). Also HW debug and flash programming I/O
11	GND	
12	VCC	

5.3 Routage

Deux versions différentes du PCB ont été tirées. La première version est pour les deux variantes fonctionnant autour d'une lampe solaire. Cette version est tirée sur un PCB rectangle et a été imprimée en deux exemplaires.

La deuxième version est conçue pour la variante prise électrique. Cette version est tirée sur un PCB rond de 52mm de diamètre. Sur le schéma électrique, seulement le connecteur I/O J1 est mis. Ceci afin d'éviter de devoir refaire le routage complet spécialement pour la variante prise électrique.

La figure 30 montre les deux versions du routage de la board nRF51822 :



Version Prise électrique

Version lampe solaire
& BQ25504

Figure 30 : Versions de la board nRF51822

Le composant nRF51822 a un pad relié à la masse qui nécessite l'utilisation d'un four.

5.4 Tests

5.4.1 Fonctionnement du nRF51822

Le connecteur de programmation fonctionne correctement, car il est possible de programmer le nRF51822.

Afin de tester cette board, deux codes exemples de nordic ont été utilisés.

Le premier test permet de valider le fonctionnement des entrées et sorties du nRF51822. Le code exemple de nordic permet d'inverser une sortie chaque seconde. Ce code fonctionne correctement. A noter que seulement les sorties utilisées ont été testées.

Le deuxième test permet de valider les connections Bluetooth Low Energy. Le code exemple de nordic contenant le « Heart Rate Service » a été adapté, puis programmé sur la plaque. Ce code exemple fonctionne correctement.

Une fois ces deux tests effectués, le programme du chapitre 4.4, programmation, a été programmé. Toutes les fonctionnalités du nRF51822 utilisées fonctionnent correctement. Les tests se trouvent dans le chapitre 4.4.

5.4.2 Consommation de la board nRF51822

Une mesure de courant de la consommation de cette board a été effectuée avec un multimètre agilent U1252A sur une gamme de mesures de 0-5000uA. La consommation est de 740uA. La mesure a été effectuée avec deux multimètres différents et le même résultat est obtenu. Cette mesure est prise sur la board nRF51822 débranchée des différentes variantes et reliée à une alimentation 3.3V. A noter que les trois board nRF51822 ont la même consommation.

Ce courant est bien plus grand que le courant attendu. L'HES-SO de Sion a fait des mesures sur le kit d'évaluation et un courant moyen de 25uA a été obtenu.

Un test a été effectué pour voir si le processeur s'endort correctement lorsque rien ne se passe. Chaque fois que le nRF51822 s'endort, la sortie passe à 1 et chaque fois qu'il se réveille elle passe à 0. Le résultat de cette mesure se trouve sur la figure 31:

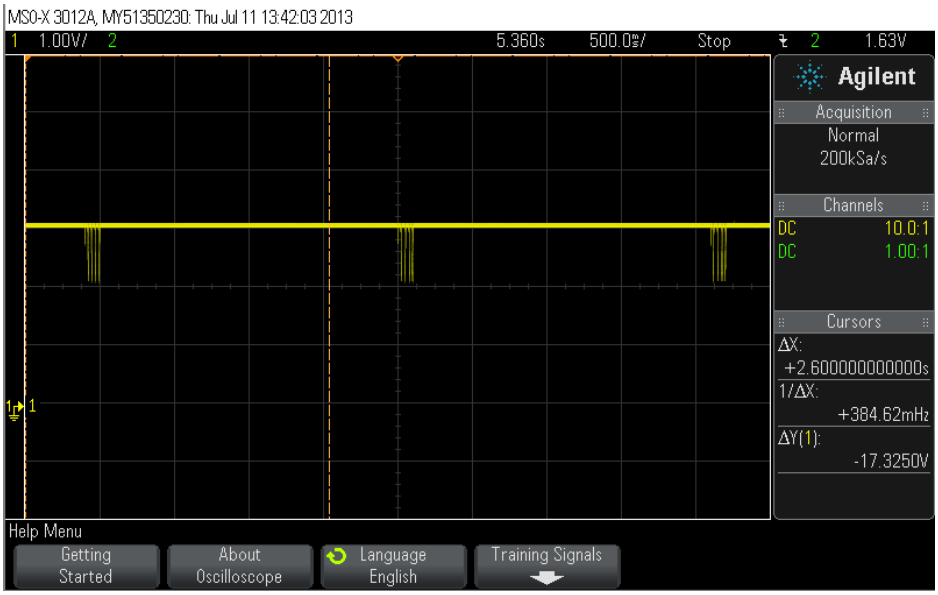


Figure 31 : Test de l'activité du nRF51822

Sur cette mesure, on peut voir que les moments oú le nRF51822 se réveille se situe au moment oú il envoie des advertise, soit toutes les deux secondes durant 100ms.

Un test avec un agilent L4532A permet de faire une mesure de courant plus précise. Une résistance de 1Ω a été placée entre l'alimentation de laboratoire et la board nRF51822. Le résultat de la mesure se situe sur la figure 32:

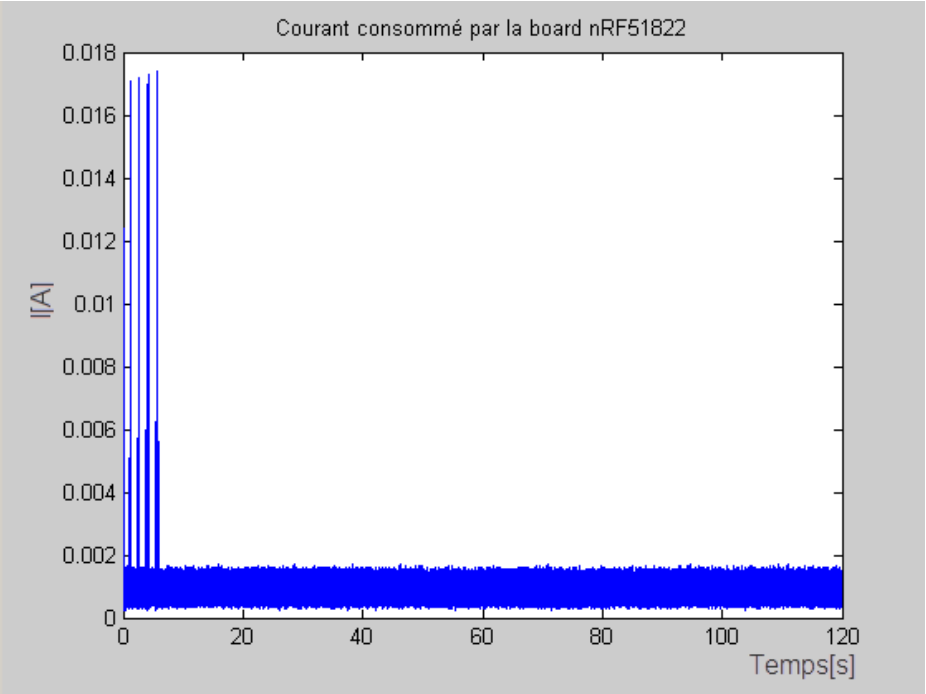


Figure 32 : mesure de courant effectuée avec un agilentL4532A

La valeur moyenne de cette mesure est de 0.878mA. Cette mesure permet de déterminer que le problème ne vient pas de l'advertiser mais de la consommation continue du circuit.

6 VARIANTE A : BOARD NRF51822 ADAPTEE A UNE LAMPE SOLAIRE

6.1 Principe

Le but de ce prototype est de piloter une lampe solaire bon marché déjà existante par Bluetooth Low Energy. Le schéma électrique de la lampe solaire achetée va donc être adapté pour y ajouter la board nRF51822.

La lampe solaire a été achetée chez Jumbo et coûte 14frs90. On peut la voir sur la figure 33 :

Figure 33 : Lampe solaire

Elle a un capteur d'obscurité et son alimentation est composée de deux accus NiMh de 600mAh de 1.2V. Elle a également un commutateur ON/OFF.

Pour cette variante, un PCB a été tiré; ce PCB permet de piloter la LED et prendre les mesures de l'accumulateur et du panneau solaire. Le schéma bloc de la figure 33 résume les liaisons entre les trois PCB.

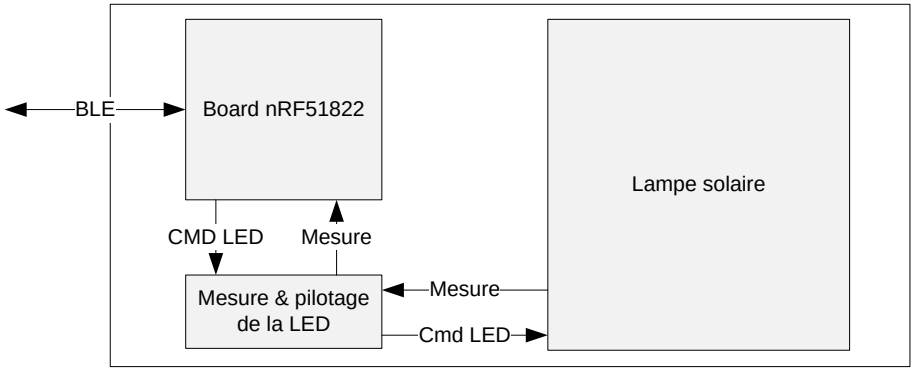


Figure 34 : schéma bloc de la variante adaptable à une lampe solaire

Le schéma électrique complet de cette variante se trouve en annexe C.

6.2 Développement

6.2.1 Schéma électrique initial de la lampe solaire

Le schéma électrique de la board qui se trouvait à l'intérieur de la lampe solaire a été relevé. La figure 35 ci-dessous le montre :

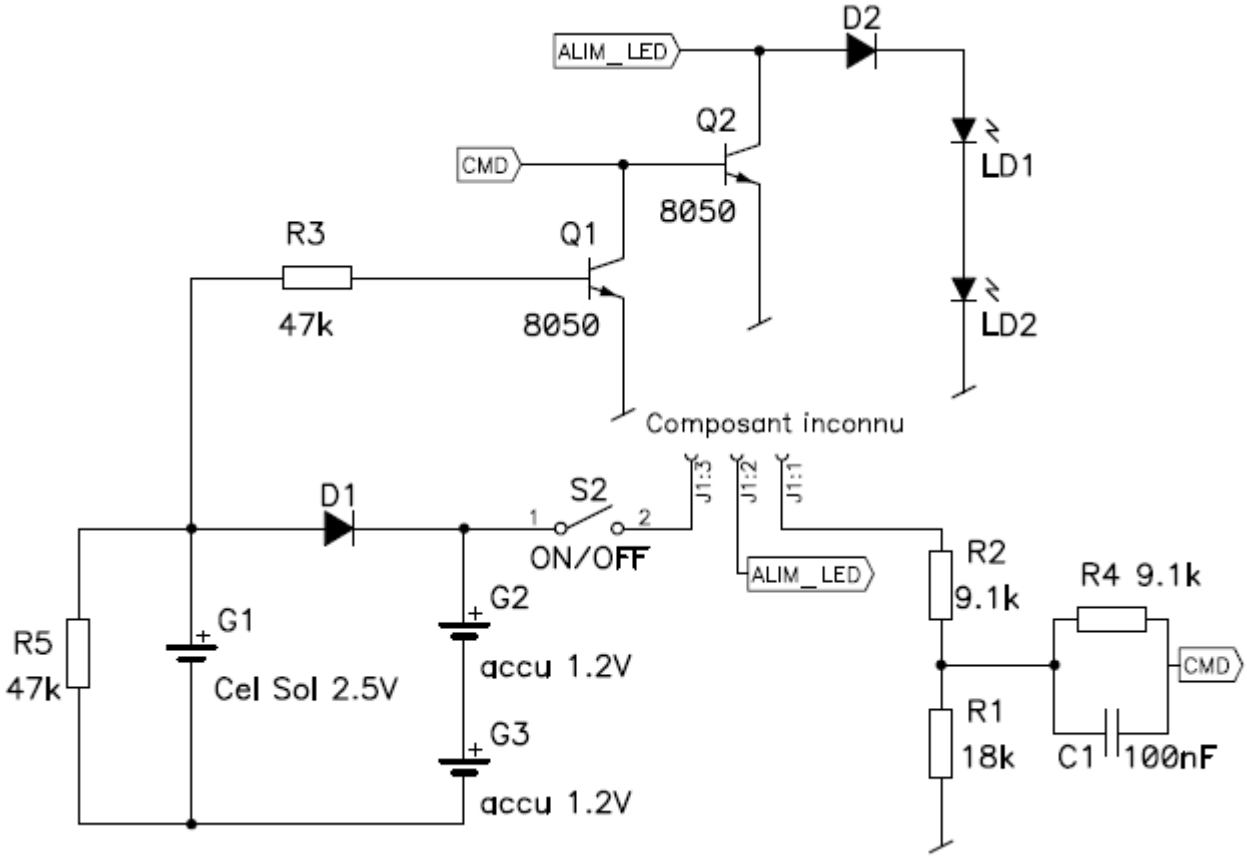


Figure 35 : Schéma électrique de la lampe solaire avant modification

La cellule solaire recharge les accumulateurs en passant par la diode D1. Le switch S2 permet d'éteindre le circuit. Lorsque le circuit est éteint, les accumulateurs vont continuer à être chargés par le panneau solaire.

Directement après le switch S2, il y a un composant dont les caractéristiques n'ont pas pu être déterminées.

La cellule solaire est reliée à Q1 par la résistance R3. Quand la cellule solaire n'a plus assez de tension pour polariser Q1, les LEDs vont s'éteindre. Cette liaison forme le capteur d'obscurité.

6.2.1 Schéma électrique avec modification de la lampe solaire

Ci-dessous le schéma électrique après modification :

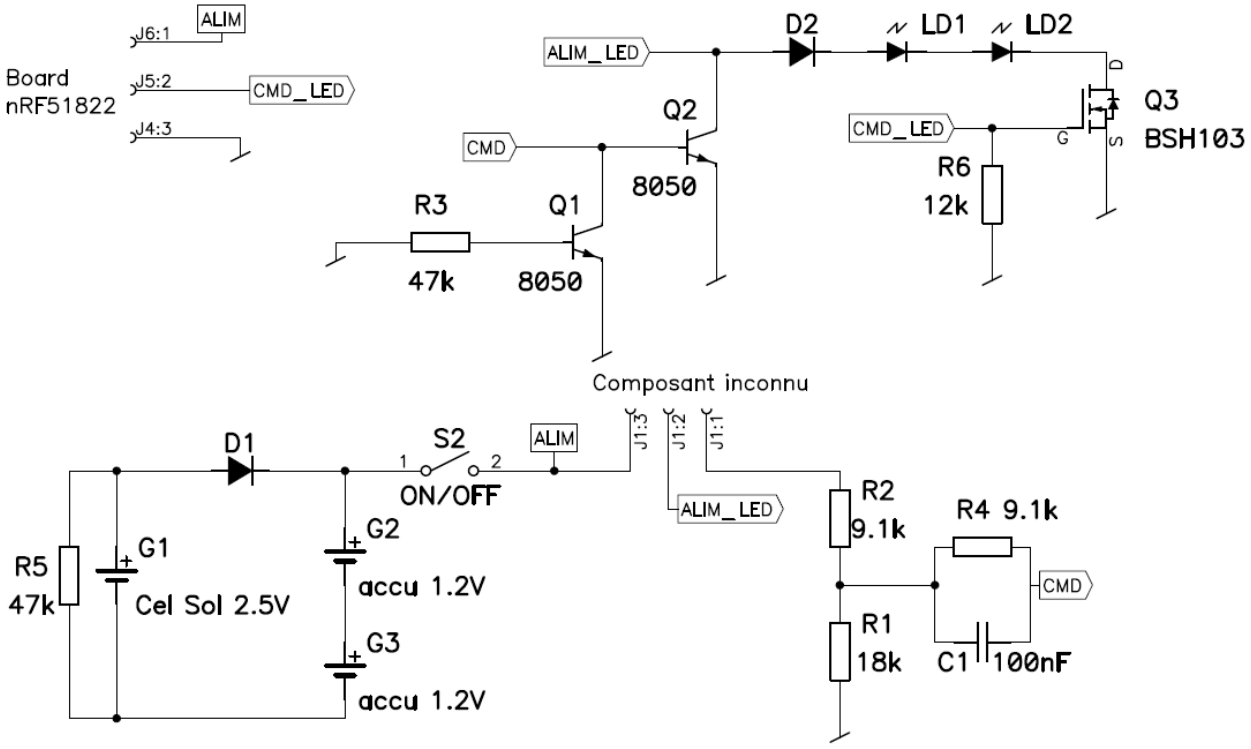


Figure 36 : Schéma électrique après modification

Deux changements ont eu lieu sur le schéma initial afin de pouvoir piloter les LEDs. Premièrement un BSH103 a été ajouté afin de piloter les LEDs LD1 et LD2 grâce au signal CMD_LED qui provient de la board du nRF51822. La résistance R6 de 12kΩ a été déterminée grâce au datasheet du nRF51822 qui conseille de mettre des résistances pull up et pull down se trouvant entre 11kΩ et 16 kΩ. A noter que la commande avec le BSH103 se situe sur la « board Mesure & pilotage de la LED ».

La deuxième modification permet de supprimer le capteur d'obscurité, car il sera implémenté dans l'application iPhone qui va couper la lampe si la tension mesurée aux bornes du panneau solaire n'est pas suffisamment élevée. Pour retirer le capteur d'obscurité, la résistance R3 est mise à la masse. Le système fonctionne comme s'il faisait toujours nuit.

Trois connexions entre la board de la lampe solaire et la board du nRF51822 existent :

- le GND
- l'alimentation
- la commande de la LED

Le nRF51822 peut fonctionner avec une tension variant de 1.8V à 3.6V. Il fonctionne donc correctement avec l'alimentation de 2.4V des deux accumulateurs. L'alimentation de la board du nRF51822 est reliée au switch S2 sur la figure 36. En éteignant le circuit à l'aide du switch S2, la board nRF51822 ne sera plus alimentée.

6.2.2 Schéma électrique de la prise de mesures du panneau solaire et des accumulateurs

La tension d'alimentation du nRF51822 n'étant pas stabilisée, la tension interne de référence de 1.2V a été utilisée pour les conversions ad. Un pont diviseur est requis, car les tensions à mesurer dépassent 1.2V.

Le même pont diviseur est appliqué sur la tension de batterie et de la cellule solaire, car les deux tensions ont la même gamme de variation. Le schéma électrique de la prise de mesure de la tension du panneau solaire et des accumulateurs se trouve sur la figure 37:

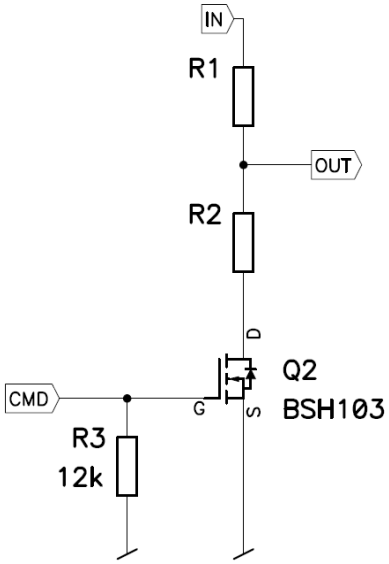


Figure 37 : Schéma électrique de la prise de mesure

L'entrée « IN » de la figure 37 est reliée à l'élément à mesurer. L'entrée « CMD » est reliée à une sortie du nRF51822 et la sortie « OUT » à un convertisseur AD du nRF51822.

Le problème du pont diviseur est qu'il consomme en permanence du courant. Le BSH103 a été rajouté afin de supprimer ce courant quand aucune mesure n'est effectuée. A noter que ce mosfet peut générer une erreur de mesure, car il a une résistance interne qui peut modifier le pont diviseur. Ce mosfet est piloté grâce à une sortie du nRF51822. La résistance R3 de 12kΩ a été déterminée grâce au datasheet du nRF51822 qui conseille de mettre des résistances pull up et pull down se trouvant entre 11kΩ et 16 kΩ.

Nordic Semiconductor a été contacté pour savoir s'il avait des valeurs prédéfinies ou des informations complémentaires au datasheet, afin de pouvoir calculer les résistances R1 et R2. Ils ont conseillé de faire des tests.

Les résistances R1 et R2 de la figure 37 ont été déterminées après une série de tests. Ces tests ont été effectués sans le BSH103 qui peut influencer les mesures. Sur le tableau 5 se trouvent les valeurs des résistances utilisées afin de déterminer R1 et R2. :

Tableau 5 : Valeur des résistances utilisées lors des tests de pont diviseur

Rtot [Ω]	126k	77k	50k	10.1k	5k
R1 [Ω]	75k	47k	30k	6.2k	3.4k
R2 [Ω]	51k	30k	20k	3.9k	1.6k

A noter que tous les ponts diviseurs testés ont un gain se rapprochant de 0.4. Ceci permet de comparer les valeurs mesurées.

Pour effectuer la mesure, une alimentation est reliée à l'entrée « IN » de la figure 37. Ensuite, la sortie « OUT » est mesurée grâce au convertisseur AD du nRF51822 qui l'envoie par Bluetooth Low Energy sur un iPad. Le tableau 6 contient les valeurs reçues par l'application lightBlue de l'iPad selon la résistance totale du pont diviseur et la tension sur l'entrée « IN » du circuit.

Tableau 6 : mesure de la valeur reçue sur l'iPad selon la résistance du pont diviseur.

Uin[V]	Val iPad [-]				
	Rtot=126kΩ	Rtot=77kΩ	Rtot=50kΩ	Rtot=10.1kΩ	Rtot=5kΩ
0	31	17	11	3	2
0.5	75	52	49	43	43
1	127	88	89	91	86
1.5	179	125	128	123	128
2	231	161	167	165	169
2.5	255	198	206	204	208

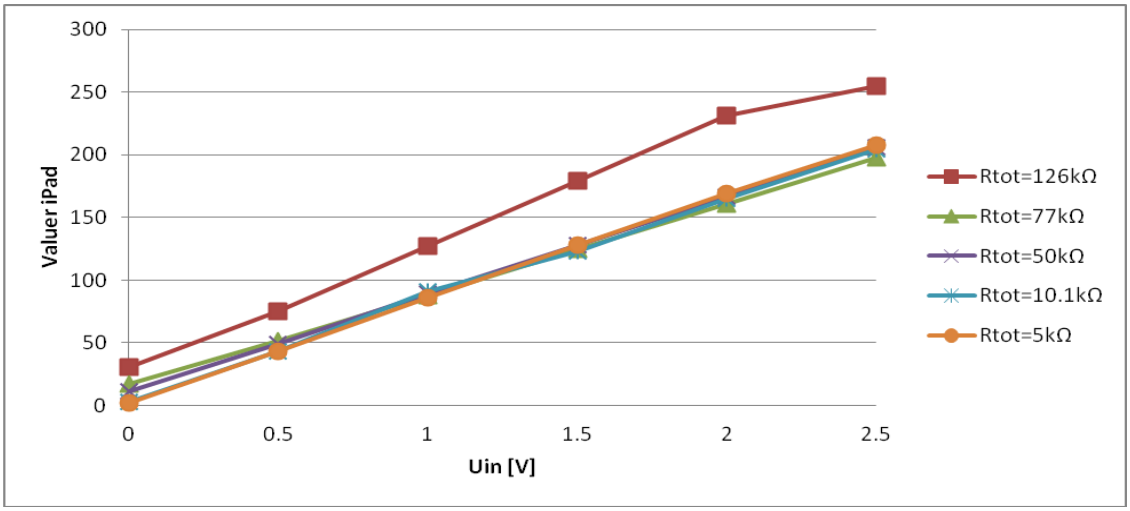


Figure 38 : graphique de la variation de la conversion AD selon la résistance du pont diviseur

Le graphique de la figure 38 montre que plus la résistance globale du pont diviseur augmente, plus la mesure d'une tension se rapprochant du 0V sera faussée.

Avec une résistance de 10kΩ, le 0V correspond à la valeur 3. Ce qui équivaut à une erreur de 14.17 mV. Une résistance globale de 10kΩ a été choisie. Ce pont résistif consommera jusqu'à 420uA lors d'une mesure.

6.3 Test

6.3.1 Mesure du RSSI

Trois mesures du RSSI selon la distance entre le serveur et le périphérique ont été faites. L'une au milieu d'une place goudronnée, la suivante au milieu d'un pré et la troisième dans le couloir d'un bâtiment. La mesure du RSSI a été effectuée à l'aide d'un iPad 2. L'application LightBlue peut mesurer le RSSI seulement si le peripheral est connecté. La mesure du RSSI sur l'application LightBlue est encadrée en rouge sur la figure 39:

Figure 39 : Prise de mesure du RSSI avec l'application LightBlue

Le résultat de la mesure obtenue se trouve sur la figure 40.

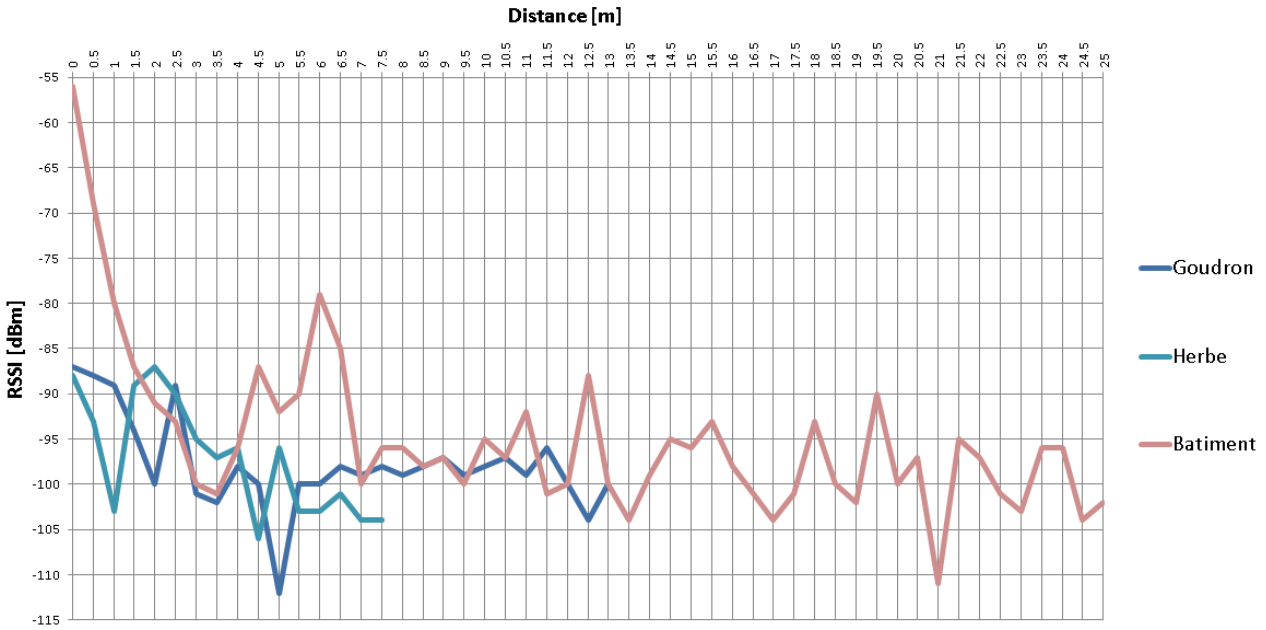


Figure 40 : Mesure du signal RSSI de la variante lampe solaire

Ce graphique montre que le signal RSSI varie beaucoup selon l'environnement dans lequel le module Bluetooth se trouve. La portée du module est deux fois plus grande dans un bâtiment que sur une place goudronnée.

Lors de la mesure, le signal RSSI n'était jamais stable. La valeur qui revenait le plus souvent durant 6 variations a été retenue.

Il faudrait essayer de faire une moyenne d'une vingtaine de valeurs à chaque emplacement afin de voir si une moyenne stable s'en détache et si l'on peut y voir une similitude entre la variation du RSSI et celle de la distance.

A noter que sur les premiers mètres, le signal RSSI varie en fonction de la distance.

7 VARIANTE B : LAMPE SOLAIRE

7.1 Principe

Dans cette variante, une lampe solaire a été développée. Elle est composée d'un panneau solaire, d'un accumulateur, d'une LED et de différents blocs permettant la commutation de la LED, la gestion de l'énergie et la connexion Bluetooth Low Energy. La lampe solaire se décompose en ces différentes parties :

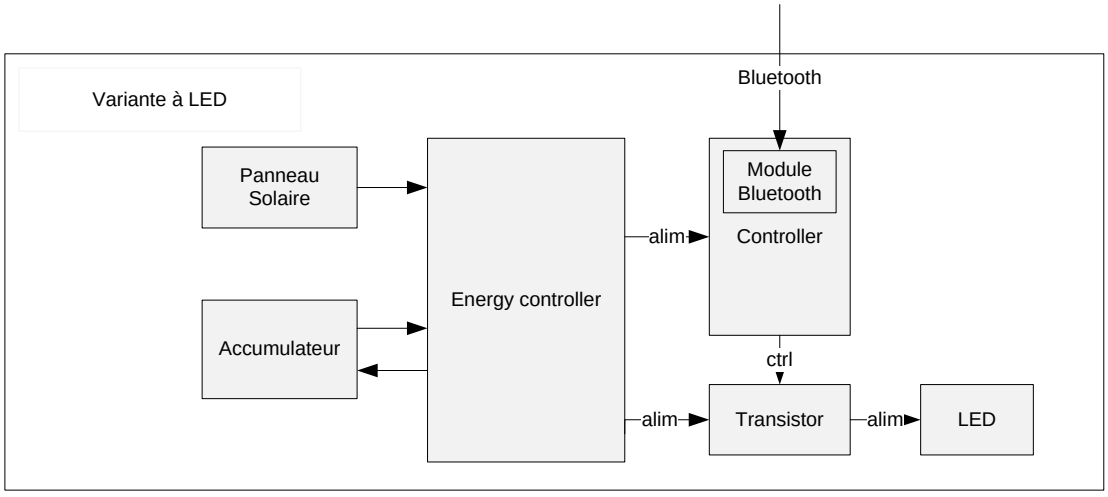


Figure 41 : Schéma de principe de la lampe solaire

Chacune de ces parties est détaillée au chapitre 7.2. Le module Bluetooth est composé de la board nRF51822 qui est détaillé au chapitre 5.

7.2 Choix des composants

7.2.1 Energy controller

Principe

Ce bloc est la constituante principale de la lampe solaire. L'énergie générée par la cellule solaire doit être stockée la journée afin d'être utilisée la nuit venue. Le cahier des charges exige l'emploi d'une seule cellule solaire. La tension de cette dernière doit être adaptée à la tension de l'accumulateur tout en surveillant le niveau de charge de ce dernier.

Pour se faire, un convertisseur DC/DC est utilisé. Ce convertisseur doit avoir des seuils de démarrage très faibles afin que le système stocke de l'énergie même si très peu de luminosité est présente sur la cellule solaire.

Une cellule solaire est un générateur dont la caractéristique est non linéaire. En conséquence, pour un même éclairement, la puissance délivrée sera différente selon la charge. Un contrôleur MPPT, Maximum Power Point Tracker, permet de piloter le convertisseur reliant l'accumulateur et la cellule solaire de manière à fournir en permanence le maximum de puissance à l'accumulateur. La figure 42 montre une courbe caractéristique d'une cellule solaire. La courbe a été prise sur Wikipedia :

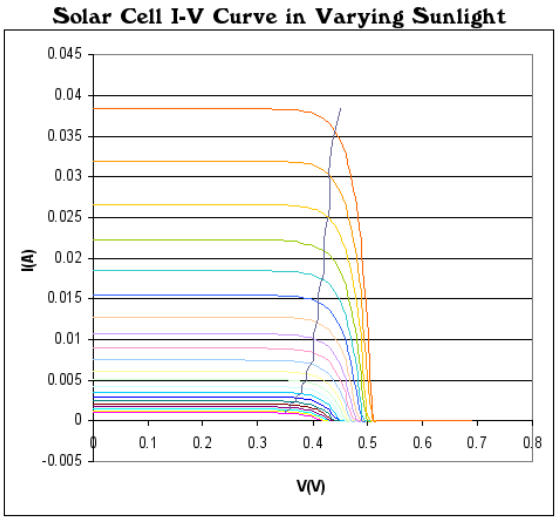


Figure 42 : courbe caractéristique d'une cellule solaire

Le convertisseur DC/DC doit inclure la possibilité d'utiliser une MPPT. Ceci permettra de stocker le maximum de puissance possible durant la journée.

Deux convertisseurs DC/DC correspondant aux exigences ont été trouvés : le bq25504 et le LTC3105.

LTC3105

Ce convertisseur DC/DC a besoin d'une tension d'entrée de 0.4V pour démarrer. Une fois lancé, la tension d'entrée peut redescendre jusqu'à 0.25V. La tension de sortie peut varier de 1.5V à 5.25V. Le LTC3105 coûte 14.35 francs/pièce (Prix chez Farnell).

Ce circuit est fait pour travailler avec des panneaux solaires. Il inclut donc une MPPT. Son gros défaut est qu'il n'a pas de sécurité pour le chargement des accumulateurs. Il utilise des accumulateurs Ni-Mh.

bq25504

Ce convertisseur DC/DC a besoin d'une tension d'entrée de 0.33V pour démarrer. Une fois lancé, la tension d'entrée peut redescendre jusqu'à 0.13V. La tension de sortie peut varier de 2.5V à 5.25V. Il coûte 6.75 francs/pièce (Prix chez Farnell). Il est donc moins cher que le LTC3105.

Ce circuit est fait pour travailler avec des panneaux solaires. Il inclut donc une MPPT. Il demande plus de composants externes que le LTC3105, mais a le gros avantage de sécuriser le chargement et le déchargement des accumulateurs. Il utilise des accumulateurs au Li-Ion.

Choix du bloc « Energie Controller »

Le tableau 7 résume les points importants qui ont été observés dans le choix du convertisseur DC/DC.

Tableau 7 : comparaison des convertisseurs DC/DC

Type	Uinmin	Uinstart	Uoutmin	Uoutmax	accu	MPPT	surveillance	prix
LTC3105	0.25V	0.4V	1.5V	5.25V	NiMh	oui	non	14.35frs
bq25504	0.13V	0.33V	2.5V	5.25V	Li-Ion	oui	oui	6.75frs

On peut voir que le bq25504 est meilleur dans la majorité des domaines. Son gros avantage est qu'il a une surveillance du chargement des accus. Il est également moins cher. Son défaut est qu'il utilise des accumulateurs Li-Ion. Ces accumulateurs sont plus chers et fonctionnent moins bien avec des températures inférieures à 0°C que les accumulateurs NiMh. En revanche, ils n'ont pas d'effet mémoire.

Le bq25504 a été choisi, car il a une surveillance de fin de chargement et de déchargement de l'accu.

Cellule solaire

Deux paramètres importants ont permis de déterminer la cellule solaire choisie : le prix et la tension nominale.

La cellule solaire doit avoir une tension nominale inférieure à 2.5 [V]. C'est la tension limite que le bq25504 peut gérer en entrée. Sur le tableau 8 se trouvent les deux cellules solaires testées.

Tableau 8 : comparaison panneau solaire

No.	fournisseur	code produit	Unom	Inom	prix	taille
1	conrad	191267 - 62	0.5V	**200 mA**	**4.55frs**	36*56 mm
2	conrad	190441 - 62	0.5V	**850 mA**	**6.75frs**	60*60 mm

Lors du branchement sur le bq25504 de la première cellule solaire, le courant en sortie de la cellule solaire pouvait monter jusqu'à 20mA. Afin d'avoir plus de puissance, le deuxième panneau solaire a été testé. Le courant que délivre ce dernier monte jusqu'à 92mA. La deuxième cellule solaire a donc été choisie.

7.2.2 Accumulateur

Le bq25504 gère le rechargement d'accu au lithium. Ces accus ont l'avantage de ne pas avoir d'effet mémoire et d'avoir un très grand nombre de cycles de rechargement. Ces accumulateurs sont toutefois plus onéreux que les Ni-Mh.

Le bq25504 peut gérer des accumulateurs dont la tension minimale vaut 2.5V. Le tableau 9 décrit les accumulateurs ayant une tension minimale supérieure à 2.5V.

Tableau 9 : comparaison des accumulateurs Li-Ion

No.	fournisseur	code produit	Unom	capacité	Energie	Sécurité	prix
1	conrad	250440 - 62	3V	**650mAh**	1.95Wh	**Non**	**16.25frs**
2	conrad	251007 - 62	3.7V	**2200mAh**	8.14Wh	**Oui**	**16.95frs**

D'autres accumulateurs existent, mais ils sont plus chers et ont une plus grande capacité que l'accumulateur No.2. Pour 70 centimes en plus, l'accumulateur No.2 a une plus grande capacité et une sécurité de déchargement. Il a donc été choisi.

7.3 Développement

Le schéma électrique complet de la variante bq25504 se trouve en annexe D.

7.3.1 Dimensionnement du bq25504

Le datasheet du bq25504 contient des exemples d'utilisation. Le schéma électrique s'inspire donc de ces exemples.

Dimensionnement des seuils de déchargement et de chargement de l'accumulateur

Le datasheet de l'accumulateur indique que la tension ne doit pas être supérieure à 4.3V et que la tension minimale ne doit pas passer en dessous de 3V.

Deux ponts diviseurs permettent de déterminer les tensions de seuils du chargement et du déchargement de l'accumulateur. Le datasheet préconise une résistance des ponts diviseurs totale de 10MΩ.

Le seuil inférieur se nomme VBAT_UV et a été défini à 3.2V. Les résistances peuvent être calculées grâce à la formule suivante :

$$VBAT\_UV = VBIAS\left(1+\frac{Ruv2}{Ruv1}\right)$$

VBIAS étant égal à 1.25V.

Avec Ruv2 à 6.04MΩ et Ruv1 à 3.83MΩ, la tension VBAT_UV est égale à 3.2212V.

Le seuil supérieur se nomme VBAT_OV et a été défini à 4.2V. Les résistances peuvent être calculées grâce à la formule suivante :

$$VBAT\_OV = \frac{3}{2}VBIAS\left(1+\frac{Rov2}{Rov1}\right)$$

VBIAS étant égal à 1.25V.

Avec Rov2 à 5.62MΩ et Rov1 à 4.42MΩ, la tension VBAT_OV est égale à 4.202V.

Dimensionnement de la MPPT

Le datasheet du bq25504 conseille de fixer la MPPT à 80% quand il est utilisé avec une cellule solaire. Un pont diviseur permet de régler la valeur de la MPPT. Le datasheet du bq25504 préconise une résistance totale du pont diviseur de 20MΩ. La formule ci-dessous permet de calculer la valeur de la MPPT :

$$MPPT(Voc) = \left(\frac{Roc1}{Roc1 + Roc2} \right)$$

Avec Roc1 à 15.62MΩ et Roc2 à 4.42MΩ, la valeur du gain est de 0.779. Ce gain correspond au gain désiré. Une résistance de 15MΩ est utilisée, car aucune résistance de 15.62MΩ n'a été trouvée sur le marché.

Dimensionnement de VBAT_OK

La tension VBAT_OK permet de régler deux tensions de seuils de fonctionnement indépendamment des réglages de surtension et de sous-tension de l'accumulateur. Les deux seuils se nomment sur le datasheet VBAT_OK et VBAT_OK_HYST, VBAT_OK étant le seuil inférieur et VBAT_OK _HYST étant le seuil supérieur. A noter que sur le datasheet du bq25504, VBAT_OK est également le nom d'une sortie.

Quand la tension de l'accumulateur se trouve entre ces deux seuils, la sortie VBAT_OK sera à l'état haut. Sinon la sortie VBAT_OK sera à 0V.

Le but de cette sortie est de piloter un FET qui coupe les circuits auxiliaires au bq25504 quand l'accumulateur se trouve en dehors de la plage VBAT_OK et VBAT_OK_HYST.

Lors de ce projet, cette option n'est pas utilisée.

7.3.2 Adaptation de l'alimentation de la board nRF51822

Le nRF51822 peut être alimenté avec une tension pouvant varier de 1.8V à 3.6V. La tension de l'accumulateur peut monter jusqu'à 4.2V. Il faut donc abaisser la tension d'alimentation de l'accumulateur.

Le nRF51822 a des picks de consommation pouvant atteindre 10mA quand il envoie des advertise. Le régulateur LT1761-3 est utilisé afin d'abaisser cette tension. C'est un régulateur LDO 3V, car le système doit fonctionner correctement quand l'accumulateur est à 3.2V. Il consomme 20uA pour fonctionner.

La board nRF51822 sera donc alimentée en 3V dans cette variante.

7.3.3 Transistor

Le transistor va commander la LED. Le MOSFET BSH103 va être utilisé. Il peut conduire un courant allant jusqu'à 650mA; il est en stock et peut être commuté grâce a du 2V. Il a été simulé et fonctionne correctement. Le résultat et le schéma de la simulation se trouvent en annexe F.

7.3.4 Dimensionnement de la mesure de la tension de l'accumulateur

La tension de l'accumulateur peut varier de 3.2V à 4.2V. Il faut donc abaisser cette tension avant de la mesurer. Pour se faire, un pont diviseur est utilisé.

Pour optimiser la consommation, il faudrait utiliser un système qui permettrait de couper le pont diviseur. Ceci pourrait se faire avec un fet à canal P. Le système expliqué au chapitre 6.2.2 se basant sur un MOSFET à cannal N ne peut pas être utilisé, car lorsque le mosfet couperait, une tension de 4.2V pourrait se trouver aux bornes du nrf51822. Cette tension n'est pas supportable pour le nRF51822 qui peut monter jusqu'à 3.6V.

Le pont diviseur a une résistance totale de 80kΩ. Il est composé d'une résistance de 20kΩ et d'une de 60kΩ. Ces résistances ont été définies de façon à avoir la tension

maximale de l'accumulateur dans la plage 0V-1.2V; 0V–1.2V étant la plage de conversion du convertisseur ad du nRF51822.

La tension aux bornes du nRF51822 peut être calculée grâce au calcul suivant (R1=20kΩ & R2=60kΩ) :

$$Uin = Uaccu * \frac{R1}{R1 + R2}$$

L'accumulateur variant de 3.2V à 4.2V, la tension aux bornes du nRF51822 va donc varier de 0.800V à 1.05V. Ces valeurs sont inférieures à 1.2V et sont donc dans la plage de conversion du nRF51822.

7.3.5 Dimensionnement de la mesure de la tension du panneau solaire

Le panneau solaire variant de 0 à 0.5V est directement relié à une entrée AD du nRF51822. Ces tensions se situent en dessous du 1.2V qui est la tension de référence interne au nRF51822 pour les conversion AD.

7.4 Routage

Trois connecteurs sont présents afin de pouvoir connecter la LED, la cellule solaire et la batterie. Un jumper permet de couper l'alimentation de la board du nRF51822, ceci afin de faire des tests.

7.5 Tests

Les différentes mesures de ce chapitre ont été effectuées à l'aide de deux multimètres Agilent U1252A.

A noter que lors des différents tests effectués, les résistances permettant la mesure de la tension de l'accumulateur ont été retirées. Ceci afin qu'elles n'interfèrent pas avec les résultats obtenus.

7.5.1 Mesure de la tension du seuil de chargement de l'accumulateur

Afin de tester si le chargement de l'accumulateur s'arrête à la bonne tension, l'accumulateur a été remplacé par une alimentation de laboratoire. Le schéma de mesure se trouve sur la figure 43 :

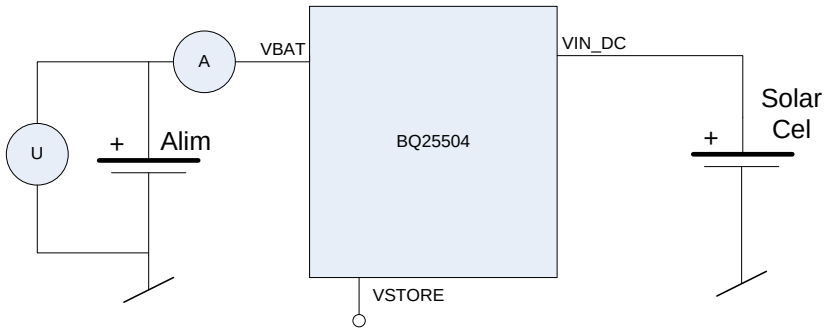


Figure 43 : Schéma de mesure de la tension maximale de l'accumulateur

Une fois le système sous tension, la tension de l'alimentation de laboratoire remplaçant l'accumulateur est augmentée jusqu'à ce que le bq25504 coupe le chargement. Le chargement est coupé lorsque l'ampèremètre indique 0mA. Le résultat de ce test se trouve sur le tableau 10.

Tableau 10 : Résultat de la mesure de la tension maximale de l'accumulateur

Ubat[V]	Iinbat[mA]
4.1997	1.2
4.1998	0

Le bq25504 coupera donc le chargement lorsque que l'accumulateur atteindra les 4.1998V. Le seuil supérieur réglé étant 4.2002V, l'erreur absolue est donc de 0.0004V.

L'erreur relative de mesure de l'Agilent U1252A pour une plage de 5V est de 0.025%. Ce chiffre a été pris sur son datasheet. Sur une mesure de 4.1998V, cela représente une erreur de 1.0499mV.

L'erreur absolue du multimètre étant plus grande que l'erreur absolue mesurée, il se peut que la tension soit supérieure à 4.2002V de quelques millivolts.

D'après le datasheet de l'accumulateur, la tension à ne pas dépasser est de 4.3V. Le bq25504 coupe donc correctement la charge de l'accumulateur.

7.5.2 Mesure de la tension du seuil de déchargement de l'accumulateur

Afin de tester si le déchargement de l'accumulateur s'arrête à la bonne tension, l'accumulateur a été remplacé par une alimentation de laboratoire. Le schéma de mesure se trouve sur la figure 44 :

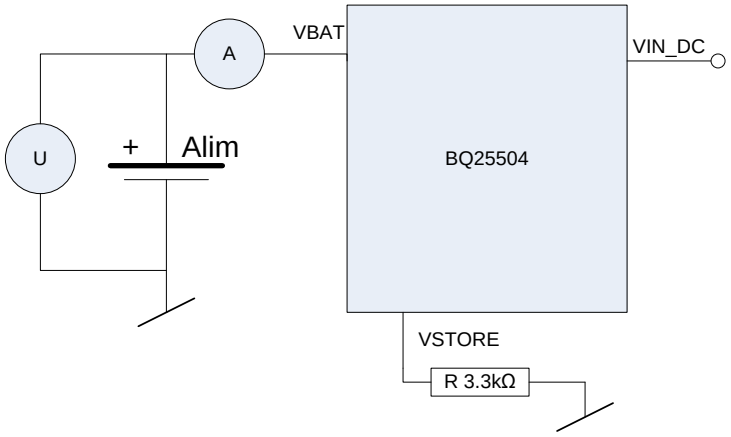


Figure 44 : Schéma de mesure de la tension maximale de l'accumulateur

Une fois le système sous tension, la tension de l'alimentation de laboratoire remplaçant l'accumulateur est diminuée jusqu'à ce que le bq25504 coupe le déchargement. Le déchargement est coupé lorsque l'ampèremètre indique 0mA. Le résultat de ce test se trouve sur le tableau 11 :

Tableau 11 : Résultat de la mesure de la tension minimale de l'accumulateur

Ubat[V]	Ioutbat[mA]
3.2290	2.64
3.2289	0

Le bq25504 coupera donc le déchargement de l'accumulateur à 3.2289V. La tension de seuil de déchargement étant réglée à 3.2212V, l'erreur absolue est donc de 7mV. Ce qui représente une erreur relative de 0.0023%.

D'après le datasheet de l'accumulateur, la tension à ne pas dépasser est de 3V. Le bq25504 coupe donc correctement la décharge de l'accumulateur.

7.5.3 Mesure de la MPPT du bq25504

Deux mesures sont effectuées afin de calculer la MPPT :

- La mesure aux bornes de la cellule solaire débranchée du circuit
- La mesure aux bornes de la cellule solaire branchée au circuit

Les deux mesures sont effectuées avec la même luminosité sur le panneau solaire. Ci-dessous (figure 45) se trouve le schéma de mesure de la MPPT :

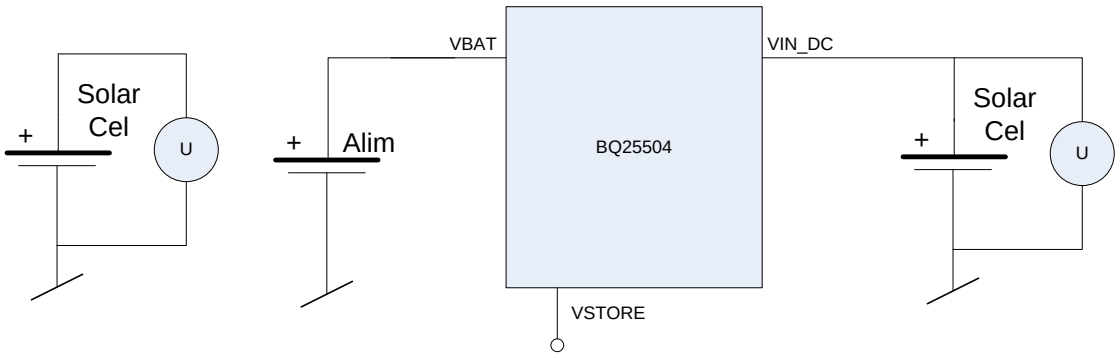


Figure 45 : schéma de la mesure de la MPPT

Le résultat de la mesure se trouve sur le tableau 12 :

Tableau 12 : résultats de la mesure de la MPPT

Tension panneau solaire		rapport [%]
sans circuit[V]	avec circuit[V]	
0.474	0.3794	80.0
0.3112	0.238	76.5
0.271	0.204	75.3
0.13	0.098	75.4
0.0154	0.012	77.9

Le rapport moyen est de 77.02%. Ce qui représente une erreur de 3% par apport au résultat attendu qui est de 80%. Cette erreur provient du fait qu'une résistance de 15MΩ a été utilisée au lieu d'une résistance de 15.62MΩ. Avec une résistance de 15 MΩ, le rapport est de 77,23%. L'erreur absolue est donc de 0.21%. Ce qui représente une erreur relative de 0.2%. La MPPT fonctionne donc correctement.

7.5.4 Mesure du chargement et déchargement de l'accumulateur

Afin de tester le déchargement et le chargement de l'accumulateur sur la durée, le programme Agilent GUI data logger est utilisé. Ce programme permet de sauver sur un fichier Excel la mesure d'un multimètre. Cette mesure est prise de façon régulière. Dans le cadre de ces tests, chaque 10 minutes une mesure est prise. Le schéma de mesure se trouve sur la figure 46 :

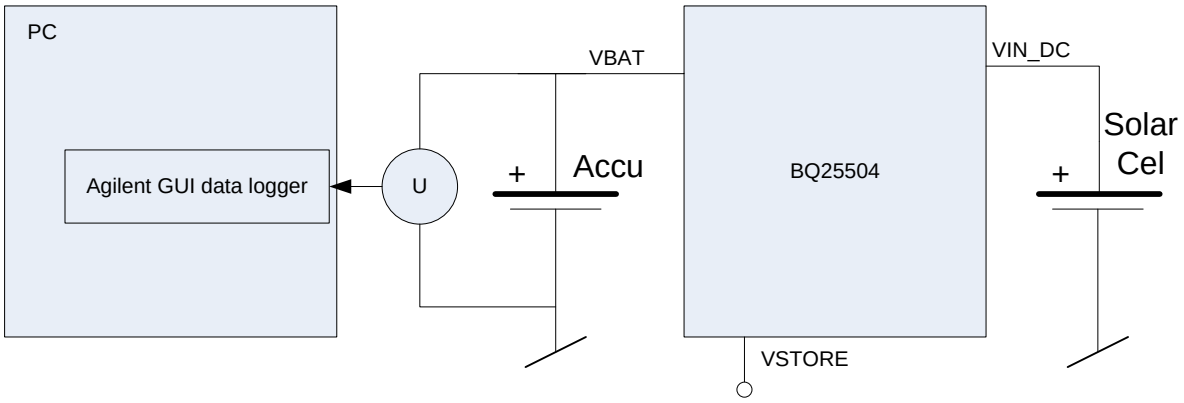


Figure 46 : schéma de mesure du chargement de l'accumulateur

Mesure du chargement de l'accumulateur

Le but de cette mesure est de tester le chargement de l'accumulateur sans charge sur la pin VSTORE du bq25504. Cette mesure a été effectuée sur 24h. La figure 47 montre le résultat obtenu :

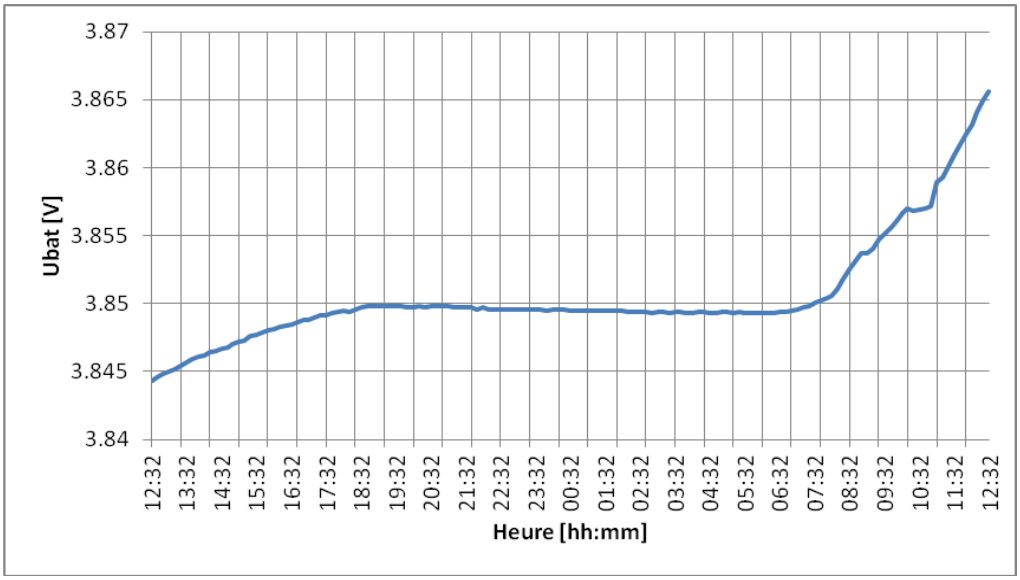


Figure 47 : Résultat de la mesure du chargement de l'accumulateur

On peut constater que l'accumulateur n'est plus chargé à partir de 18h30. Ceci est dû au fait que les stores du bâtiment A de l'HES-SO se ferment automatiquement à 18h30.

La tension de l'accumulateur a tout de même augmenté de 25mV en 24h. On peut voir également que la pente de la courbe est nettement plus élevée le deuxième jour. Cette variation est due à l'ensoleillement qui était plus faible le premier jour.

Mesure du déchargement de l'accumulateur avec la board nRF51822

Le but de cette mesure est de tester le déchargement de l'accumulateur lorsque peu de courant est tiré. Le panneau solaire est donc débranché. La board nRF51822 est branchée à la pin VSTORE du bq25504. Cette mesure a été effectuée sur 24h. La figure 48 montre le résultat obtenu :

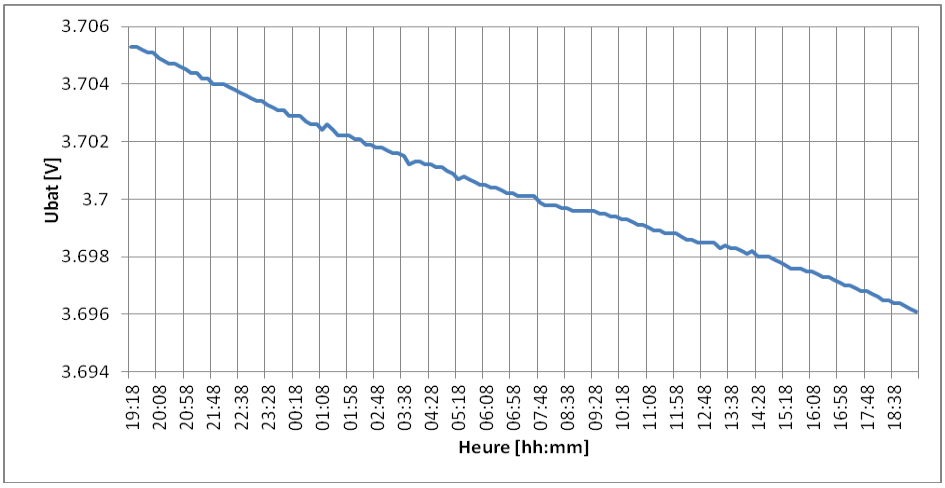


Figure 48 : mesure du déchargement de l'accumulateur avec la board nRF51822

Cette mesure permet de valider le fonctionnement du bq25504 quand le courant minimal de la variante lampe solaire sera tiré.

Mesure du déchargement de l'accumulateur avec une led allumée

Dans cette mesure, le panneau solaire et la board nRF51822 sont débranchés. Un fils relie Vstore au mosfet qui permet d'allumer la LED. Cette mesure a été effectuée sur 24h. La figure 49 montre le résultat obtenu :

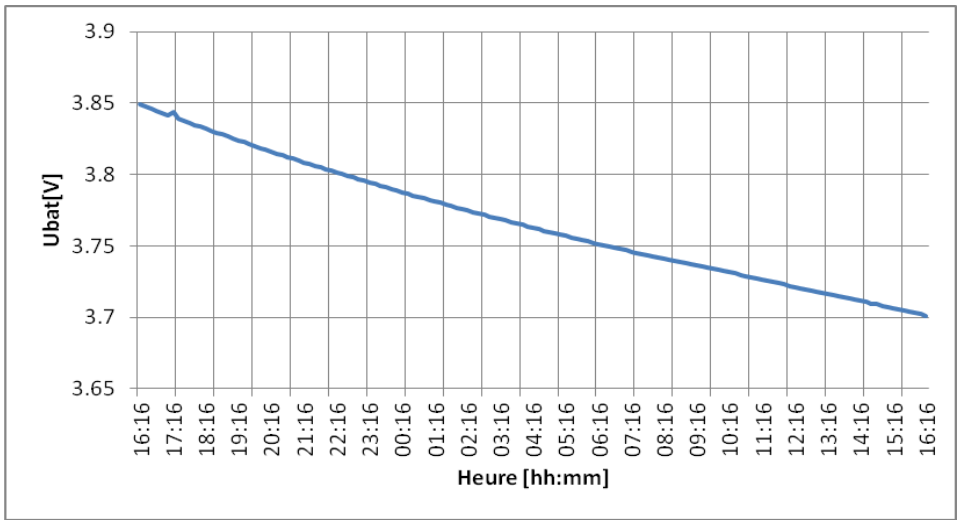


Figure 49 : mesure du déchargemnet de l'accumulateur avec une Led allumée

Cette mesure permet de valider le fonctionnement du bq25504 quand le courant maximal de la variante lampe solaire est tiré.

7.5.5 RSSI mesure

Trois mesures du RSSI selon la distance entre la variante lampe solaire et un iPad ont été faites. L'une au milieu d'une place goudronnée, la suivante sur un pré et la troisième dans le couloir d'un bâtiment. La mesure du RSSI a été effectuée à l'aide d'un iPad. A noter que l'application LightBlue peut mesurer le RSSI seulement si le périphérique est connecté. La figure 50 montre le résultat obtenu.

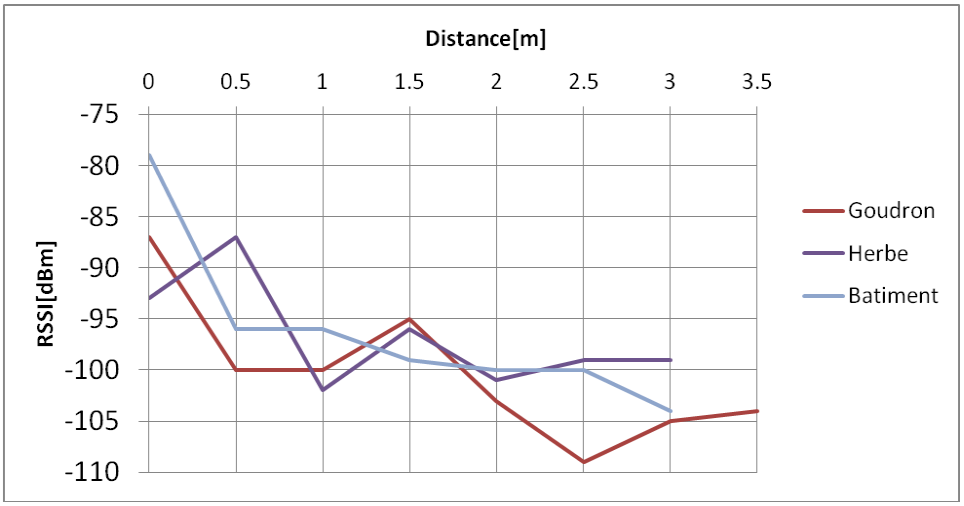


Figure 50 : Mesure du signal RSSI de la variante lampe solaire

Ce graphique montre que cette variante n'est plus détectée à partir de 3m50. Ceci peut venir de l'adaptation d'impédance qui n'est pas optimisée. Afin de l'optimiser, il faudrait utiliser un balun pour faire l'adaptation d'impédance.

Le RSSI a tendance à diminuer selon la distance, mais la mesure est effectuée sur 3m50. Il faut donc améliorer l'adaptation d'impédance et refaire la mesure sur une plus grande distance.

8 VARIANTE C : PRISE MURALE

8.1 Principe

Dans cette variante, une prise murale pilotée par Bluetooth a été développée. Cette variante se décompose en ces différentes parties :

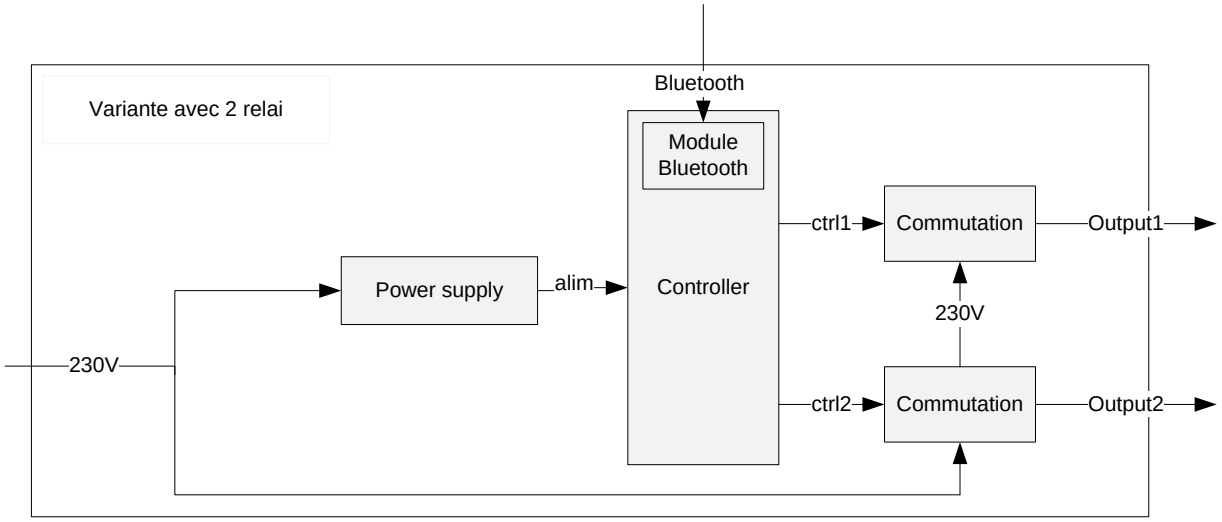


Figure 51 : Schéma de principe de la prise murale

Cette variante est divisée donc en 3 parties, la partie module Bluetooth, la partie power supply et la partie commutation. Les deux dernières parties sont détaillées dans le chapitre 8.2. Le module Bluetooth est détaillé au chapitre 5 de ce rapport.

8.2 Choix des composants

8.2.1 Partie commutation

Principe

Ce bloc est la constituante principale du système. Ce dernier devant remplacer une prise réseau quelconque, la charge est, dès lors, totalement inconnue.

Le but est donc de commuter une charge de type capacitive, inductive ou résistive consommant au maximum 10A nominal sur un réseau de puissance, dans ce cas le 230V.

Une attention particulière doit être apportée sur le plan de l'échauffement, car le milieu est clos et ne permet aucune dissipation.

Elément de commutation

Il existe différents éléments de commutation pour le courant :

- Triacs
- Relais
- Relais statique
- COOLMOS

Les critères de sélection sont le volume, la simplicité d'utilisation et l'échauffement.

Avec un triac, chaque état (actif ou inactif) doit être répété continuellement afin d'éviter les enclenchements intempestifs.

Il reste donc le COOLMOS, le relais et le relais statique. Le gros avantage du relais statique est qu'il peut avoir le détecteur de passage par zéro inclus dans le boitier, mais malheureusement leur résistance interne est très grande. Par exemple le SKL20220 a une tension de 19V à ses bornes quand on tire un courant de 10A. Ce qui fait qu'il dissipe 190W.

Sur Farnel, les plus faibles résistances internes des COOLMOS tournent autour des 100mΩ. Ce qui est plus grand que la résistance des relais. La résistance de certains relais qui se trouvaient dans les stocks de l'HES-SO a été mesurée et se situe aux alentours des 40mΩ. L'avantage des COOLMOS est qu'ils prennent moins de place que les relais, mais leur prix est plus élevé.

Le tableau ci-dessous résume les avantages et les inconvénients des différents éléments de commutation :

Tableau 13 : récapitulatif des différents éléments de commutation

	Avantages	Désavantages
Relais	Peu d'échauffement Commande simple	Durée de vie à pleine charge
Diacs, triacs	Dimension	Echauffement Coûts
COOLMOS	Dimension	Echauffement Coûts
Relais statique	Détection de passage par 0	Echauffement Coûts

Le relais a été choisi. Il existe deux types de relais utilisables, le monostable et le bistable. Le relais bistable consomme moins de courant, mais est généralement plus grand et plus cher. Dans le cadre de ce projet, deux relais monostables sont utilisés.

Choix du relais

Le tableau 14 résume les différents relais trouvés :

Tableau 14 : comparaison des relais

N°	fournisseur	No	type	Ucmd	U AC nom	Ron	Inom	Pdis	dimension	Prix/pce
1	Farnell	2060856	bistable	12V	250V	3	16	300	12.7*29	9.45
2	Farnell	2060857	bistable	5V	250V	3	16	300	12.7*15.7	10.9
3	Farnell	1891815	monostable	3V	240V	0.1	16	10	12.8*20	3.5
4	Farnell	2060804	monostable	5V	250V	0.04	16	3.5	12.7*15.7	4.65

Dans ce tableau, la puissance dissipée est calculée pour un courant traversant le relais de 10A.

La majorité des *datasheet* sur les relais indique comme résistance interne 100mΩ max. De ce fait, on ne connaît pas précisément leur résistance. Le relais n°4 a été choisi. Il a la résistance interne la plus faible. A noter qu'il dissipe 3.5W lorsqu'il tire 10A. Sa résistance interne a été mesurée et est de 40mΩ.

8.2.2 Power supply

Principe

En reprenant les chapitres 5 et 8.2.1, le système nécessite les alimentations suivantes :

- +5V : relais
- +3.3V : module Bluetooth

Le système nécessite donc deux tensions continues différentes.

Observons maintenant les charges de chaque partie :

- +5V : ~45mA : Relais
- +3.3V : ~10mA (pick) : Module Bluetooth advertiser
  ~25uA : Module Bluetooth alimentation

Une séparation galvanique n'est pas forcément nécessaire, car le système sera entièrement isolé de l'utilisateur.

Plusieurs solutions existent :

- Transformateur
- Module de type traco
- Alimentation de type FlyBack

Le défaut du transformateur est qu'il prend de la place. L'alimentation de type FlyBack est plus complexe à développer. L'optimisation de la partie hardware n'étant pas une priorité, le traco a été choisi.

Le traco choisi est le TMLM04253. Il convertit la tension du secteur (90V à 264V) en une tension continue de 5V et de 3.3V. Il peut fournir jusqu'à 600mA sur le 5V et 150mA sur le 3.3V. Le tout pour 21.50 francs (Farnell). Il fait 36.5mm de long et 27mm de large. Il prendra donc la grande partie de la place disponible.

Une évolution future serait d'utiliser une alimentation de type fly-back. Elle serait plus compacte que le traco.

8.3 Routage

Le routage doit passer dans un cercle de 52mm. De ce fait, il y a trois étages. Sur le premier se trouvent les relais ainsi que les connexions vers le 230V. Sur le deuxième se trouve le traco et le troisième est constitué de la board nRF51822 détaillée au chapitre 5. Le PCB total fait 50mm de hauteur. Il serait possible de diminuer cette dimension en remplaçant le traco par un autre type de convertisseur AC/DC.

8.4 Mécanique

Pour ce prototype, un boitier de démonstration a été produit. Le boitier a été réalisé grâce à un tube et à des pièces plastiques. Le boitier a été conçu sur Autodesk Inventor. Les pièces en plastique ont été réalisées grâce à une découpeuse laser de l'HES-SO. Elles font toutes 1.6mm d'épaisseur. La figure 52 représente une vue en coupe du boiter de cette variante. La pièce verte représente le PCB.

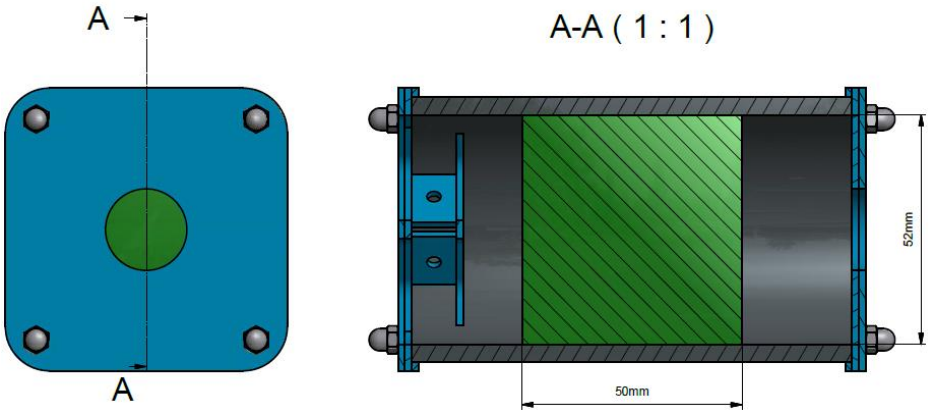


Figure 52 : coupe du prototype de la prise murale

Dans le tube, de la place a été laissée pour les fils ainsi que pour les deux prises qui ne sont pas sur cette figure. A noter que le tube fait 120mm de long, mais que toute la place n'est pas utilisée. Cette longueur pourrait être 0réduite. La figure 53 montre l'éclaté du boitier:

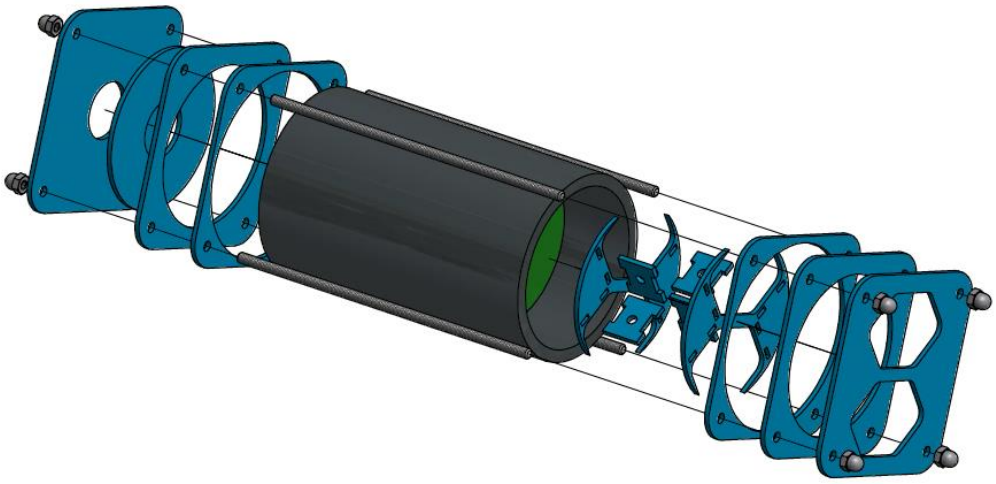


Figure 53 : éclaté du prototype de la prise murale

Le système devant être démontable, dans un premier temps, les deux facettes les plus proches du tube devaient être fixées avec de la colle. La dernière facette est vissée aux facettes collées. Cela permet de fixer le système. Au final, des tiges filetées sont utilisées. De ce fait, la facette du milieu est devenue inutile. Cette pièce est tout de même présente sur le prototype, ceci afin d'éviter de devoir retirer toutes les facettes.

8.5 Tests

La board nRF51822 a tout d'abords été programmée grâce à une alimentation de laboratoire. Une fois le protocole mis sous tension, mesuré et validé, la board nRF51822 a été branchée au reste du circuit.

Le test effectué sur ce prototype consiste juste à allumer et à éteindre des lampes de bureau alimentées en 230V. Les lampes s'allument correctement.

9 EVOLUTION FUTURE

Pour la suite du projet, certaines choses doivent être effectuées :

- Déterminer pourquoi la board nRF51822 consomme 740uA.
- Refaire un print de la version lampe solaire contenant une commande pour la mesure de l'accumulateur.
- Faire une mesure du RSSI plus poussée afin d'en déterminer un rapport avec la distance.
- Utiliser un balun pour l'adaptation d'impédance de l'antenne, ceci afin de stabiliser la portée des différentes variantes.
- Tester si la variante prise murale peut conduire 10A en prêtant attention au dégagement de chaleur.
- Faire une alimentation basée sur le principe flyback pour la version prise murale.
- Faire une détection de passage par zéro pour la commutation des relais de la version prise murale. Ceci afin d'améliorer leurs durées de vie.
- Développer une application iPhone selon l'étude de cas d'utilisation.

10 CONCLUSION

Le développement de la carte du nRF51822 a été réalisé. Le nRF51822 peut être programmé et tous les tests effectués sur la programmation ont été un succès. Toutes les fonctionnalités du nRF51822 utilisées dans le cadre de ce projet fonctionnent. Toutefois, il consomme plus de courant que la consommation attendue.

Pour la variante adaptable sur une lampe solaire, les mesures de la tension des deux accumulateurs et de la cellule solaire ont des erreurs relatives suffisamment faibles pour permettre d'avoir un aperçu convenable des tensions à leurs bornes. La lampe se pilote correctement.

Dans la variante lampe solaire, les seuils de sécurité de l'accumulateur sont précis. La MPPT fonctionne correctement. Le bq25504 a réagi convenablement à tous les tests effectués.

La dernière variante permet de piloter du 230V. Les relais s'enclenchent correctement. Cependant la longueur de la variante reste trop grande pour rentrer dans une prise murale.

A l'avenir, il serait intéressant de faire plus de mesures du RSSI pour pouvoir développer une application iPhone.

Sion le 12 juillet 2013

11 REFERENCE

11.1 Site internet

http://nordicsemi.com

http://bluetooth.com

http://agilent.com

http://distrelec.ch

http://mauser.com

http://farnell.com

12 TABLE DES FIGURES

13 TABLE DES TABLEAUX

14 ANNEXE

Annexe A: Estimation des coûts & liste des composants

Annexe B: Schematic : Board nRF51822

Annexe C: Schematic : Variante adaptable à une lampe solaire

Annexe D: Schematic : Variante lampe solaire

Annexe E: Schematic : Variante prise murale

Annexe F: Simulation BSH103

Annexe G: Code C : service MMS

Annexe H: Code C : service BAS

Annexe I: Code C : service CSS

Annexe J: Code C : Variante adaptable à une lampe solaire

Annexe K: Code C : Variante lampe solaire

Annexe L: Code C : Variante prise murale

Annexe M: Logiciel de développement

Estimation des coûts

Les tableaux ci-dessous résument les coûts pour la fabrication d'une pièce par prototype sans le botier.

Estimation des coûts des différentes boards

Board nRF51822			
Quantité	Désignation	prix unité	total
1	QUARTZ 16 MHZ 10 PF CMS	3.25	3.25
1	EMBASE 2 R 10V 1.27mm	1.85	1.85
1	nRF51822	5.00	5.00
1	composants divers	5.00	5.00
		total	15.10

Variante lampe solaire			
Quantité	Désignation	prix unité	total
1	Lampe solaire	14.90	14.90
2	BSH103	0.58	1.16
1	composants divers	5.00	5.00
1	board nRF51822	15.10	15.10
		total	36.16

Variante bq25504			
Quantité	Désignation	prix unité	total
1	bq25504	5.52	5.52
1	regulateur 3V LDO	2.60	2.60
1	Accu LiIon 3,7 V 2200 mAh	16.95	16.95
1	Cellule solaire 0.5 V 850mA	6.75	6.75
1	bobines, condensateur, résistance pres	5.00	5.00
1	composants divers	10.00	10.00
1	board nRF51822	15.10	15.10
		total	61.92

Variante prise murale			
Quantité	Désignation	prix unité	total
2	relai 5V 10A	4.65	9.30
1	Traco power 230VAC -> 5V-3.3V TMLM 04253	23.10	23.10
3	Barrette mâle droite 2P 5 mm	1.51	4.53
3	Barrette femelle 2P 5 mm	2.92	8.76
2	BSH103	0.58	1.16
1	composants divers	5.00	5.00
1	board nRF51822	15.10	15.10
		total	66.95

```
/* Copyright (c) 2012 Nordic Semiconductor. All Rights Reserved.
 *
 * The information contained herein is property of Nordic Semiconductor ASA.
 * Terms and conditions of usage are described in detail in NORDIC
 * SEMICONDUCTOR STANDARD SOFTWARE LICENSE AGREEMENT.
 *
 * Licensees are granted free, non-transferable use of the information. NO
 * WARRANTY of ANY KIND is provided. This heading must NOT be removed from
 * the file.
 *
 */

/** @file
 *
 * @defgroup ble_sdk_app_template_main main.c
 * @{
 * @ingroup ble_sdk_app_template
 * @brief Template project main file.
 *
 * This file contains a template for creating a new application. It has the
code necessary to wakeup from button, advertise, get a connection
 * restart advertising on disconnect and if no new connection created go
back to system-off mode.
 * It can easily be used as a starting point for creating a new application,
the comments identified with 'YOUR_JOB' indicates where
 * and how you can customize.
 */

#include <stdint.h>
#include <string.h>

#include "nrf.h"
#include "app_error.h"
#include "nrf_gpio.h"
#include "nrf51_bitfields.h"
#include "boards/pca10001.h"
#include "ble.h"
#include "ble_hci.h"
#include "ble_srv_common.h"
#include "ble_advdata.h"
#include "ble_conn_params.h"
#include "ble_stack_handler.h"
#include "ble_radio_notification.h"
#include "ble_flash.h"
#include "ble_debug_assert_handler.h"
#include "ble_error_log.h"

#include "app_timer.h"
#include "app_gpiote.h"
#include "app_scheduler.h"

#include "nordic_common.h"
#include "ble_mms.h"


#define DEVICE_NAME                     "Pres_VLAMPE"
/**< Name of device. Will be included in the advertising data. */
#define MANUFACTURER_NAME               "HES-SO Valais"
/**< Manufacturer. Will be passed to Device Information Service. */

#define APP_ADV_INTERVAL                32
```



```
/**< The advertising interval (in units of 0.625 ms. This value corresponds
to 1 s). (32=20ms)*/
#define APP_ADV_TIMEOUT_IN_SECONDS
BLE_GAP_ADV_TIMEOUT_GENERAL_UNLIMITED       /**< The advertising timeout (in
units of seconds). */
#define NBR_OF_ADVERTISE_IN_BURST               5
#define BASE_TIME_INTERVAL                          625

#define SECOND_1_25_MS_UNITS            800
/**< Definition of 1 second, when 1 unit is 1.25 ms. */
#define SECOND_10_MS_UNITS              100
/**< Definition of 1 second, when 1 unit is 10 ms. */
#define MIN_CONN_INTERVAL               (SECOND_1_25_MS_UNITS / 2)
/**< Minimum acceptable connection interval (0.5 seconds), Connection
interval uses 1.25 ms units. */
#define MAX_CONN_INTERVAL               (SECOND_1_25_MS_UNITS)
/**< Maximum acceptable connection interval (1 second), Connection interval
uses 1.25 ms units. */
#define SLAVE_LATENCY                   0
/**< Slave latency. */
#define CONN_SUP_TIMEOUT                (4 * SECOND_10_MS_UNITS)
/**< Connection supervisory timeout (4 seconds), Supervision Timeout uses 10
ms units. */

#define FIRST_CONN_PARAMS_UPDATE_DELAY  APP_TIMER_TICKS(5000,
APP_TIMER_PRESCALER) /**< Time from initiating event (connect or start of
notification) to first time sd_ble_gap_conn_param_update is called (15
seconds). */
#define NEXT_CONN_PARAMS_UPDATE_DELAY   APP_TIMER_TICKS(5000,
APP_TIMER_PRESCALER)  /**< Time between each call to
sd_ble_gap_conn_param_update after the first (5 seconds). */
#define MAX_CONN_PARAMS_UPDATE_COUNT    3
/**< Number of attempts before giving up the connection parameter
negotiation. */

#define APP_GPIOTE_MAX_USERS            1
/**< Maximum number of users of the GPIOTE handler. */

#define SEC_PARAM_TIMEOUT               30
/**< Timeout for Pairing Request or Security Request (in seconds). */
#define SEC_PARAM_BOND                  1
/**< Perform bonding. */
#define SEC_PARAM_MITM                  0
/**< Man In The Middle protection not required. */
#define SEC_PARAM_IO_CAPABILITIES       BLE_GAP_IO_CAPS_NONE
/**< No I/O capabilities. */
#define SEC_PARAM_OOB                   0
/**< Out Of Band data not available. */
#define SEC_PARAM_MIN_KEY_SIZE          7
/**< Minimum encryption key size. */
#define SEC_PARAM_MAX_KEY_SIZE          16
/**< Maximum encryption key size. */

#define DEAD_BEEF                       0xDEADBEEF
/**< Value used as error code on stack dump, can be used to identify stack
location on stack unwind. */
```

```
#define APP_TIMER_PRESCALER             0
/**< Value of the RTC1 PRESCALER register. */
#define APP_TIMER_MAX_TIMERS            4
/**< Maximum number of simultaneously created timers. */
#define APP_TIMER_OP_QUEUE_SIZE         5
/**< Size of timer operation queues. */


#define ADVERTISER_TIMER_INTERVAL_LENGTH    APP_TIMER_TICKS(2000,
APP_TIMER_PRESCALER) /**< timer for send the burst of advertise interval
(ticks). */
#define ADVERTISER_TIMER_INTERVAL_SHORT
APP_TIMER_TICKS(NBR_OF_ADVERTISE_IN_BURST*APP_ADV_INTERVAL*BASE_TIME_INTERVA
L/1000, APP_TIMER_PRESCALER) /**< timer for send the burst of advertise
interval (ticks). */
#define OUTPUT_OFF_TIMER_INTERVAL                   APP_TIMER_TICKS(4000,
APP_TIMER_PRESCALER) /**< timer for solar LED interval (ticks). */


#define SCHED_MAX_EVENT_DATA_SIZE       sizeof(app_timer_event_t)
/**< Maximum size of scheduler events. Note that scheduler BLE stack events
do not contain any data, as the events are being pulled from the stack in
the event handler. */
#define SCHED_QUEUE_SIZE                10
/**< Maximum number of events in the scheduler queue. */

#define MODE_ADV_WAIT                                   0
/**< constant when the advertiser is on the wait mode. */
#define MODE_ADV_ADVERTISE                          1
/**< constant when the advertiser advertise*/


#define OUTPUT1                                         LED0
/**< Is on when we are in Mild Alert state. */
#define OUTPUT2                                         LED1
/**< Is on when we are in Mild Alert state. */



static  uint16_t stateOfAdvertise;




static  app_timer_id_t                  m_outPut_off_timer_id;
/**< timer for shutdown the output before no connection */
static  app_timer_id_t
m_advertiser_timer_id;                                              /**<
advertiser timer*/

static ble_gap_sec_params_t             m_sec_params;
/**< Security requirements for this application. */
static uint16_t                         m_conn_handle =
BLE_CONN_HANDLE_INVALID;    /**< Handle of the current connection. */

static ble_mms_t                      m_mms;
/**< Structure used to identify the Immediate Alert service. */



static void changeOutPut(void);
```



```
static void on_mms_evt(ble_mms_t * p_mms, ble_mms_evt_t * p_evt);
static void advertising_start(void);




typedef enum
{
  TIMER0 = 0,  /*!< Timer0 module, base address at 0x40008000 */
  TIMER1,      /*!< Timer1 module, base address at 0x40009000 */
  TIMER2       /*!< Timer2 module, base address at 0x4000A000 */
} timer_t;

/**@brief Error handler function, which is called when an error has
occurred.
 *
 * @warning This handler is an example only and does not fit a final
product. You need to analyze
 *          how your product is supposed to react in case of error.
 *
 * @param[in] error_code  Error code supplied to the handler.
 * @param[in] line_num    Line number where the handler is called.
 * @param[in] p_file_name Pointer to the file name.
 */
void app_error_handler(uint32_t error_code, uint32_t line_num, const uint8_t
* p_file_name)
{
    NVIC_SystemReset();
}


/**@brief Assert macro callback function.
 *
 * @details This function will be called in case of an assert in the
SoftDevice.
 *
 * @warning This handler is an example only and does not fit a final
product. You need to analyze
 *          how your product is supposed to react in case of Assert.
 * @warning On assert from the SoftDevice, the system can only recover on
reset.
 *
 * @param[in]   line_num   Line number of the failing ASSERT call.
 * @param[in]   file_name  File name of the failing ASSERT call.
 */
void assert_nrf_callback(uint16_t line_num, const uint8_t * p_file_name)
{
    app_error_handler(DEAD_BEEF, line_num, p_file_name);
}


/**@brief LEDs initialization.
 *
 * @details Initializes all LEDs used by the application.
 */
static void leds_init(void)
{
      GPIO_LED_CONFIG(OUTPUT1);
        GPIO_LED_CONFIG(OUTPUT2);
        nrf_gpio_pin_clear(OUTPUT1);
```

```
        nrf_gpio_pin_clear(OUTPUT2);
}

/*****************************************************************************
**
* Static Timeout Handling Functions
*****************************************************************************
*/


/**@brief advertiser timer timeout handler.
 *
 * @details This function will be called each time advertiser timer expires.
 *
 * @param[in]   p_context   Pointer used for passing some arbitrary
information (context) from the
 *                          app_start_timer() call to the timeout handler.
 */
static void time_advertiser_timeout_handler(void * p_context)
{
    uint16_t err_code;


    if(stateOfAdvertise==MODE_ADV_ADVERTISE){
        err_code = sd_ble_gap_adv_stop();   //stop advertise
    APP_ERROR_CHECK(err_code);
        stateOfAdvertise=MODE_ADV_WAIT;     //change state
        err_code = app_timer_stop(m_advertiser_timer_id);
        APP_ERROR_CHECK(err_code);
        err_code = app_timer_start(m_advertiser_timer_id,

ADVERTISER_TIMER_INTERVAL_LENGTH,
                                                            NULL);
        APP_ERROR_CHECK(err_code);
    }
    else
            advertising_start();                    //start advertise
    //start timer advertise
            err_code = app_timer_stop(m_advertiser_timer_id);
            err_code = app_timer_start(m_advertiser_timer_id,

ADVERTISER_TIMER_INTERVAL_SHORT,
                                                            NULL);
            APP_ERROR_CHECK(err_code);
}



/**@brief shut down Output timer timeout handler.
 *
 * @details This function will be called each time solar LED timer expires.
 *
 * @param[in]   p_context   Pointer used for passing some arbitrary
information (context) from the
 *                          app_start_timer() call to the timeout handler.
 */
static void time_outPut_off_timeout_handler(void * p_context)
{
    UNUSED_PARAMETER(p_context);
        if (m_mms.mode==MODE_AUTOMATIQUE)
        {
```



```
            m_mms.outPut=OUTPUT_OFF;
            changeOutPut();
        }
}

/**@brief Timer initialization.
 *
 * @details Initializes the timer module.
 */
static void timers_init(void)
{
    // Initialize timer module, making it use the scheduler
    APP_TIMER_INIT(APP_TIMER_PRESCALER, APP_TIMER_MAX_TIMERS,
APP_TIMER_OP_QUEUE_SIZE, false);

    //Create timers
    uint32_t err_code = app_timer_create(&m_outPut_off_timer_id,
                                APP_TIMER_MODE_SINGLE_SHOT,
                                time_outPut_off_timeout_handler);
    APP_ERROR_CHECK(err_code);

            //Create timers
    err_code = app_timer_create(&m_advertiser_timer_id,
                                APP_TIMER_MODE_SINGLE_SHOT,
                                time_advertiser_timeout_handler);
    APP_ERROR_CHECK(err_code);


}


/**@brief GAP initialization.
 *
 * @details This function shall be used to setup all the necessary GAP
(Generic Access Profile)
 *          parameters of the device. It also sets the permissions and
appearance.
 */
static void gap_params_init(void)
{
    uint32_t                err_code;
    ble_gap_conn_params_t   gap_conn_params;
    ble_gap_conn_sec_mode_t sec_mode;

    BLE_GAP_CONN_SEC_MODE_SET_OPEN(&sec_mode);

    err_code = sd_ble_gap_device_name_set(&sec_mode, DEVICE_NAME,
strlen(DEVICE_NAME));
    APP_ERROR_CHECK(err_code);


    err_code = sd_ble_gap_appearance_set(BLE_APPEARANCE_GENERIC_TAG);
    APP_ERROR_CHECK(err_code);

    memset(&gap_conn_params, 0, sizeof(gap_conn_params));

    gap_conn_params.min_conn_interval = MIN_CONN_INTERVAL;
    gap_conn_params.max_conn_interval = MAX_CONN_INTERVAL;
    gap_conn_params.slave_latency     = SLAVE_LATENCY;
    gap_conn_params.conn_sup_timeout  = CONN_SUP_TIMEOUT;
```

```
    err_code = sd_ble_gap_ppcp_set(&gap_conn_params);
    APP_ERROR_CHECK(err_code);
}


/**@brief Advertising functionality initialization.
 *
 * @details Encodes the required advertising data and passes it to the
stack.
 *          Also builds a structure to be passed to the stack when starting
advertising.
 */
/**@brief Advertising functionality initialization.
 *
 * @details Encodes the required advertising data and passes it to the
stack.
 *          Also builds a structure to be passed to the stack when starting
advertising.
 */
static void advertising_init(void)
{
    uint32_t      err_code;
    ble_advdata_t advdata;
    uint8_t       flags = BLE_GAP_ADV_FLAGS_LE_ONLY_GENERAL_DISC_MODE;
        stateOfAdvertise=MODE_ADV_WAIT;
    ble_uuid_t adv_uuids[] =
    {
        {BLE_UUID_HEART_RATE_SERVICE,         BLE_UUID_TYPE_BLE}
    };

    // Build and set advertising data
    memset(&advdata, 0, sizeof(advdata));

    advdata.name_type               = BLE_ADVDATA_FULL_NAME;
    advdata.include_appearance      = true;
    advdata.flags.size              = sizeof(flags);
    advdata.flags.p_data            = &flags;
    advdata.uuids_complete.uuid_cnt = sizeof(adv_uuids) /
sizeof(adv_uuids[0]);
    advdata.uuids_complete.p_uuids  = adv_uuids;

    err_code = ble_advdata_set(&advdata, NULL);
    APP_ERROR_CHECK(err_code);
}



/**@brief Initialize Radio Notification event handler.
 */
static void radio_notification_init(void)
{
    uint32_t err_code;

    err_code = ble_radio_notification_init(NRF_APP_PRIORITY_HIGH,

NRF_RADIO_NOTIFICATION_DISTANCE_4560US,
                                           ble_flash_on_radio_active_evt);
    APP_ERROR_CHECK(err_code);
}
/**@brief Initialize mms Service.
 */
```



```
static void mms_init(void)
{
    uint32_t       err_code;
    ble_mms_init_t mms_init_obj;
        m_mms.mode = MODE_AUTOMATIQUE;
    memset(&mms_init_obj, 0, sizeof(mms_init_obj));
    mms_init_obj.evt_handler = on_mms_evt;

    err_code = ble_mms_init(&m_mms , &mms_init_obj);
    APP_ERROR_CHECK(err_code);
}



/**@brief Initialize services that will be used by the application.
 */
static void services_init(void)
{

    //initialize mms Service
        mms_init();

}


/**@brief Initialize security parameters.
 */
static void sec_params_init(void)
{
    m_sec_params.timeout      = SEC_PARAM_TIMEOUT;
    m_sec_params.bond         = SEC_PARAM_BOND;
    m_sec_params.mitm         = SEC_PARAM_MITM;
    m_sec_params.io_caps      = SEC_PARAM_IO_CAPABILITIES;
    m_sec_params.oob          = SEC_PARAM_OOB;
    m_sec_params.min_key_size = SEC_PARAM_MIN_KEY_SIZE;
    m_sec_params.max_key_size = SEC_PARAM_MAX_KEY_SIZE;
}

/**@brief Connection Parameters Module handler.
 *
 * @details This function will be called for all events in the Connection
Parameters Module which
 *          are passed to the application.
 *          @note All this function does is to disconnect. This could have
been done by simply
 *                setting the disconnect_on_fail config parameter, but
instead we use the event
 *                handler mechanism to demonstrate its use.
 *
 * @param[in]   p_evt   Event received from the Connection Parameters
Module.
 */
static void on_conn_params_evt(ble_conn_params_evt_t * p_evt)
{
    uint32_t err_code;

    APP_ERROR_CHECK_BOOL(p_evt->evt_type == BLE_CONN_PARAMS_EVT_FAILED);

    err_code = sd_ble_gap_disconnect(m_conn_handle,
BLE_HCI_CONN_INTERVAL_UNACCEPTABLE);
    APP_ERROR_CHECK(err_code);
```

```
}


/**@brief Connection Parameters module error handler.
 *
 * @param[in]   nrf_error   Error code containing information about what
went wrong.
 */
static void conn_params_error_handler(uint32_t nrf_error)
{
    APP_ERROR_HANDLER(nrf_error);
}


/**@brief Initialize the Connection Parameters module.
 */
static void conn_params_init(void)
{
    uint32_t               err_code;
    ble_conn_params_init_t cp_init;

    memset(&cp_init, 0, sizeof(cp_init));

    cp_init.p_conn_params                  = NULL;
    cp_init.first_conn_params_update_delay = FIRST_CONN_PARAMS_UPDATE_DELAY;
    cp_init.next_conn_params_update_delay  = NEXT_CONN_PARAMS_UPDATE_DELAY;
    cp_init.max_conn_params_update_count   = MAX_CONN_PARAMS_UPDATE_COUNT;
    cp_init.start_on_notify_cccd_handle    = BLE_GATT_HANDLE_INVALID;
    cp_init.disconnect_on_fail             = false;
    cp_init.evt_handler                    = on_conn_params_evt;
    cp_init.error_handler                  = conn_params_error_handler;

    err_code = ble_conn_params_init(&cp_init);
    APP_ERROR_CHECK(err_code);
}



/**@brief Start advertising.
 */
static void advertising_start(void)
{
    uint32_t             err_code;
    ble_gap_adv_params_t adv_params;

        stateOfAdvertise=MODE_ADV_ADVERTISE;
    // Start advertising
    memset(&adv_params, 0, sizeof(adv_params));

    adv_params.type        = BLE_GAP_ADV_TYPE_ADV_IND;
    adv_params.p_peer_addr = NULL;
    adv_params.fp          = BLE_GAP_ADV_FP_ANY;
    adv_params.interval    = APP_ADV_INTERVAL;
    adv_params.timeout     = APP_ADV_TIMEOUT_IN_SECONDS;

    err_code = sd_ble_gap_adv_start(&adv_params);
    APP_ERROR_CHECK(err_code);

}
```



```

/**@brief Immediate Alert event handler.
 *
 * @details This function will be called for all Immediate Alert events
which are passed to the
 *          application.
 *
 * @param[in]   p_ias  Immediate Alert stucture.
 * @param[in]   p_evt  Event received from the Immediate Alert service.
 */
static void on_mms_evt(ble_mms_t * p_mms, ble_mms_evt_t * p_evt)
{
    switch (p_evt->evt_type)
    {
        case BLE_MMS_EVT_OUTPUT_UPDATED:
            //alert_signal(p_evt->params.alert_level);
            break;

        default:
            break;
    }
}



/**@brief Application's BLE Stack event handler.
 *
 * @param[in]   p_ble_evt   Bluetooth stack event.
 */
static void on_ble_evt(ble_evt_t * p_ble_evt)
{
    uint32_t                         err_code = NRF_SUCCESS;
    static ble_gap_evt_auth_status_t m_auth_status;
    ble_gap_enc_info_t *             p_enc_info;

    switch (p_ble_evt->header.evt_id)
    {
        case BLE_GAP_EVT_CONNECTED:

                        //stop timer for advertise
                        err_code = app_timer_stop(m_advertiser_timer_id);
                        APP_ERROR_CHECK(err_code);

                        //stop timer for shut down outPut
                        err_code = app_timer_stop(m_outPut_off_timer_id);
                        APP_ERROR_CHECK(err_code);


            break;

        case BLE_GAP_EVT_DISCONNECTED:
           if (m_mms.mode==MODE_AUTOMATIQUE)
                                    {
                                        //change output
                                        m_mms.outPut=OUTPUT1_ON;
                                        changeOutPut();
                                        // Start detection timer.
                                        err_code =
app_timer_start(m_outPut_off_timer_id,

OUTPUT_OFF_TIMER_INTERVAL,
```

```
NULL);
                                    APP_ERROR_CHECK(err_code);
                                }
                                //change mode on automatique of manuel
finish
                        else if((m_mms.mode
==MODE_MANUEL)&&(m_mms.outPut==0x00))
                                {
                                    m_mms.mode=MODE_AUTOMATIQUE;
                                }
                        m_conn_handle = BLE_CONN_HANDLE_INVALID;

                        if (err_code == NRF_SUCCESS)
                            {

                                //start timer advertise
                                stateOfAdvertise=MODE_ADV_WAIT;
                                err_code =
app_timer_start(m_advertiser_timer_id,

ADVERTISER_TIMER_INTERVAL_SHORT,

NULL);
                                APP_ERROR_CHECK(err_code);




                            }
                        break;

        case BLE_GAP_EVT_SEC_PARAMS_REQUEST:
            err_code = sd_ble_gap_sec_params_reply(m_conn_handle,

BLE_GAP_SEC_STATUS_SUCCESS,
                                                   &m_sec_params);
            break;

        case BLE_GATTS_EVT_SYS_ATTR_MISSING:
            err_code = sd_ble_gatts_sys_attr_set(m_conn_handle, NULL, 0);
            break;

        case BLE_GAP_EVT_AUTH_STATUS:
            m_auth_status = p_ble_evt->evt.gap_evt.params.auth_status;
            break;

        case BLE_GAP_EVT_SEC_INFO_REQUEST:
            p_enc_info = &m_auth_status.periph_keys.enc_info;
            if (p_enc_info->div ==
p_ble_evt->evt.gap_evt.params.sec_info_request.div)
            {
                err_code = sd_ble_gap_sec_info_reply(m_conn_handle,
p_enc_info, NULL);
            }
            else
            {
                // No keys found for this device
                err_code = sd_ble_gap_sec_info_reply(m_conn_handle, NULL,
NULL);
            }
```



```
            break;

        case BLE_GAP_EVT_TIMEOUT:

            break;

        default:
            break;
    }
    APP_ERROR_CHECK(err_code);
}



/**@brief Dispatches a BLE stack event to all modules with a BLE stack event
handler.
 *
 * @details This function is called from the scheduler in the main loop
after a BLE stack
 *          event has been received.
 *
 * @param[in]   p_ble_evt   Bluetooth stack event.
 */
static void ble_evt_dispatch(ble_evt_t * p_ble_evt)
{
        on_ble_evt(p_ble_evt);
    ble_conn_params_on_ble_evt(p_ble_evt);
        ble_mms_on_ble_evt(&m_mms, p_ble_evt);
}


/**@brief BLE stack initialization.
 *
 * @details Initializes the SoftDevice and the BLE event interrupt.
 */
static void ble_stack_init(void)
{
    BLE_STACK_HANDLER_INIT(  NRF_CLOCK_LFCLKSRC_SYNTH_250_PPM,
                             BLE_L2CAP_MTU_DEF,
                             ble_evt_dispatch,
                             true);

}
/**@brief Event Scheduler initialization.
 */
static void scheduler_init(void)
{
    APP_SCHED_INIT(SCHED_MAX_EVENT_DATA_SIZE, SCHED_QUEUE_SIZE);
}

/**@brief Initialize GPIOTE handler module.
 */
static void gpiote_init(void)
{
    APP_GPIOTE_INIT(APP_GPIOTE_MAX_USERS);
}



/**@brief Power manager.
```

```
 */
static void power_manage(void)
{
    uint32_t err_code = sd_app_event_wait();
    APP_ERROR_CHECK(err_code);
}
/**@brief change the value of the output.
 */
static void changeOutPut(void)
{
        switch (m_mms.outPut){
                    case OUTPUT1_ON:    nrf_gpio_pin_set(OUTPUT1);

nrf_gpio_pin_clear(OUTPUT2);
                                                    break;
                    case OUTPUT2_ON:    nrf_gpio_pin_clear(OUTPUT1);

nrf_gpio_pin_set(OUTPUT2);
                                                    break;
                    case OUTPUT12_ON:   nrf_gpio_pin_set(OUTPUT1);

nrf_gpio_pin_set(OUTPUT2);
                                                    break;
                    case OUTPUT_OFF:    nrf_gpio_pin_clear(OUTPUT1);

nrf_gpio_pin_clear(OUTPUT2);
                                                    break;
                    default:                    nrf_gpio_pin_clear(OUTPUT1);

nrf_gpio_pin_clear(OUTPUT2);
                                                    break;
            }
}



/**@brief Application main function.
 */
int main(void)
{
    // Initialize
    leds_init();
    gpiote_init();
    ble_stack_init();
        ble_error_log_init();
      timers_init();
      scheduler_init();
    gap_params_init();
    advertising_init();
    services_init();
    conn_params_init();
    sec_params_init();
        radio_notification_init();
    m_mms.outPut=OUTPUT_OFF;
        m_mms.mode=MODE_AUTOMATIQUE;
    //start timer advertise
        uint16_t err_code;
        err_code = app_timer_start(m_advertiser_timer_id,

ADVERTISER_TIMER_INTERVAL_LENGTH,
                                                        NULL);
```



```
        APP_ERROR_CHECK(err_code);

    // Enter main loop
    for (;;)
    {
            app_sched_execute();
            changeOutPut();
      power_manage();
        }
}

/**
 * @}
 */
```

```
/* Copyright (c) 2012 Nordic Semiconductor. All Rights Reserved.
 *
 * The information contained herein is property of Nordic Semiconductor ASA.
 * Terms and conditions of usage are described in detail in NORDIC
 * SEMICONDUCTOR STANDARD SOFTWARE LICENSE AGREEMENT.
 *
 * Licensees are granted free, non-transferable use of the information. NO
 * WARRANTY of ANY KIND is provided. This heading must NOT be removed from
 * the file.
 *
 */

/** @file
 *
 * @defgroup ble_sdk_app_template_main main.c
 * @{
 * @ingroup ble_sdk_app_template
 * @brief Template project main file.
 *
 * This file contains a template for creating a new application. It has the
code necessary to wakeup from button, advertise, get a connection
 * restart advertising on disconnect and if no new connection created go
back to system-off mode.
 * It can easily be used as a starting point for creating a new application,
the comments identified with 'YOUR_JOB' indicates where
 * and how you can customize.
 */

#include <stdint.h>
#include <string.h>

#include "nrf.h"
#include "app_error.h"
#include "nrf_gpio.h"
#include "nrf51_bitfields.h"
#include "boards/pca10001.h"
#include "ble.h"
#include "ble_hci.h"
#include "ble_srv_common.h"
#include "ble_advdata.h"
#include "ble_conn_params.h"
#include "ble_stack_handler.h"
#include "ble_radio_notification.h"
#include "ble_flash.h"
#include "ble_debug_assert_handler.h"
#include "ble_error_log.h"

#include "app_timer.h"
#include "app_gpiote.h"
#include "app_scheduler.h"

#include "nordic_common.h"
#include "ble_mms.h"
#include "ble_bas.h"
#include "ble_css.h"

#define DEVICE_NAME                     "Pres_VBQ"
/**< Name of device. Will be included in the advertising data. */
#define MANUFACTURER_NAME               "HES-SO Valais"
/**< Manufacturer. Will be passed to Device Information Service. */
```



```
#define APP_ADV_INTERVAL                32
/**< The advertising interval (in units of 0.625 ms. This value corresponds
to 1 s). (32=20ms)*/
#define APP_ADV_TIMEOUT_IN_SECONDS
BLE_GAP_ADV_TIMEOUT_GENERAL_UNLIMITED       /**< The advertising timeout (in
units of seconds). */
#define NBR_OF_ADVERTISE_IN_BURST               5
#define BASE_TIME_INTERVAL                          625

#define SECOND_1_25_MS_UNITS            800
/**< Definition of 1 second, when 1 unit is 1.25 ms. */
#define SECOND_10_MS_UNITS              100
/**< Definition of 1 second, when 1 unit is 10 ms. */
#define MIN_CONN_INTERVAL               (SECOND_1_25_MS_UNITS / 2)
/**< Minimum acceptable connection interval (0.5 seconds), Connection
interval uses 1.25 ms units. */
#define MAX_CONN_INTERVAL               (SECOND_1_25_MS_UNITS)
/**< Maximum acceptable connection interval (1 second), Connection interval
uses 1.25 ms units. */
#define SLAVE_LATENCY                   0
/**< Slave latency. */
#define CONN_SUP_TIMEOUT                (4 * SECOND_10_MS_UNITS)
/**< Connection supervisory timeout (4 seconds), Supervision Timeout uses 10
ms units. */

#define FIRST_CONN_PARAMS_UPDATE_DELAY  APP_TIMER_TICKS(5000,
APP_TIMER_PRESCALER) /**< Time from initiating event (connect or start of
notification) to first time sd_ble_gap_conn_param_update is called (15
seconds). */
#define NEXT_CONN_PARAMS_UPDATE_DELAY   APP_TIMER_TICKS(5000,
APP_TIMER_PRESCALER)  /**< Time between each call to
sd_ble_gap_conn_param_update after the first (5 seconds). */
#define MAX_CONN_PARAMS_UPDATE_COUNT    3
/**< Number of attempts before giving up the connection parameter
negotiation. */

#define APP_GPIOTE_MAX_USERS            1
/**< Maximum number of users of the GPIOTE handler. */

#define SEC_PARAM_TIMEOUT               30
/**< Timeout for Pairing Request or Security Request (in seconds). */
#define SEC_PARAM_BOND                  1
/**< Perform bonding. */
#define SEC_PARAM_MITM                  0
/**< Man In The Middle protection not required. */
#define SEC_PARAM_IO_CAPABILITIES       BLE_GAP_IO_CAPS_NONE
/**< No I/O capabilities. */
#define SEC_PARAM_OOB                   0
/**< Out Of Band data not available. */
#define SEC_PARAM_MIN_KEY_SIZE          7
/**< Minimum encryption key size. */
#define SEC_PARAM_MAX_KEY_SIZE          16
/**< Maximum encryption key size. */

#define DEAD_BEEF                       0xDEADBEEF
/**< Value used as error code on stack dump, can be used to identify stack
location on stack unwind. */
```

```
#define APP_TIMER_PRESCALER             0
/**< Value of the RTC1 PRESCALER register. */
#define APP_TIMER_MAX_TIMERS            4
/**< Maximum number of simultaneously created timers. */
#define APP_TIMER_OP_QUEUE_SIZE         5
/**< Size of timer operation queues. */


#define ADCCONVERT_LEVEL_MEAS_INTERVAL          APP_TIMER_TICKS(30000,
APP_TIMER_PRESCALER) /**< Battery level measurement interval (ticks). */
#define ADVERTISER_TIMER_INTERVAL_LENGTH    APP_TIMER_TICKS(2000,
APP_TIMER_PRESCALER) /**< timer for send the burst of advertise interval
(ticks). */
#define ADVERTISER_TIMER_INTERVAL_SHORT
APP_TIMER_TICKS(NBR_OF_ADVERTISE_IN_BURST*APP_ADV_INTERVAL*BASE_TIME_INTERVA
L/1000, APP_TIMER_PRESCALER) /**< timer for send the burst of advertise
interval (ticks). */
#define OUTPUT_OFF_TIMER_INTERVAL                   APP_TIMER_TICKS(4000,
APP_TIMER_PRESCALER) /**< timer for solar LED interval (ticks). */


#define SCHED_MAX_EVENT_DATA_SIZE       sizeof(app_timer_event_t)
/**< Maximum size of scheduler events. Note that scheduler BLE stack events
do not contain any data, as the events are being pulled from the stack in
the event handler. */
#define SCHED_QUEUE_SIZE                10
/**< Maximum number of events in the scheduler queue. */

#define MODE_ADV_WAIT                                   0
#define MODE_ADV_ADVERTISE                          1

#define OUTPUT1                                         LED0
/**< Is on when we are in Mild Alert state. */
#define OUTPUT2                                         LED1
/**< Is on when we are in Mild Alert state. */

#define ADC_MEAS_TYPE_BATTERY                       0
#define ADC_MEAS_TYPE_CELLULE_SOLAR         1

#define FACTOR_CELL_SOL_LVL_IN_MILLIVOLT        1.2/254*100
#define FACTOR_BATTERY_LVL_IN_MILLIVOLT  1.2/(254*0.25)*100/2

#define INPUT_CELL_SOL_MEAS     ADC_CONFIG_PSEL_AnalogInput0
#define INPUT_BATTERY_MEAS      ADC_CONFIG_PSEL_AnalogInput1

static  uint16_t typeOfMeasAdc;  //define if the next adc is for the battery
or the cellule solar
static  uint16_t stateOfAdvertise;



static  app_timer_id_t                  m_outPut_off_timer_id;
/**< timer for shutdown the output before no connection */
static  app_timer_id_t                                  m_adc_timer_id;
/**< Battery timer. */
static  app_timer_id_t
```



```
m_advertiser_timer_id;                                                  /**<
advertiser timer*/

static ble_gap_sec_params_t             m_sec_params;
/**< Security requirements for this application. */
static uint16_t                         m_conn_handle =
BLE_CONN_HANDLE_INVALID;    /**< Handle of the current connection. */

static ble_mms_t                       m_mms;
/**< Structure used to identify the Immediate Alert service. */
static ble_bas_t                        m_bas;
/**< Structure used to identify the battery service. */
static ble_css_t                        m_css;
/**< Structure used to identify the cellule solar service. */

static void battery_start(void);
static void cellule_solar_start(void);
static void changeOutPut(void);
static void on_mms_evt(ble_mms_t * p_mms, ble_mms_evt_t * p_evt);
static void advertising_start(void);



typedef enum
{
  TIMER0 = 0,  /*!< Timer0 module, base address at 0x40008000 */
  TIMER1,      /*!< Timer1 module, base address at 0x40009000 */
  TIMER2       /*!< Timer2 module, base address at 0x4000A000 */
} timer_t;

/**@brief Error handler function, which is called when an error has
occurred.
 *
 * @warning This handler is an example only and does not fit a final
product. You need to analyze
 *          how your product is supposed to react in case of error.
 *
 * @param[in] error_code  Error code supplied to the handler.
 * @param[in] line_num    Line number where the handler is called.
 * @param[in] p_file_name Pointer to the file name.
 */
void app_error_handler(uint32_t error_code, uint32_t line_num, const uint8_t
* p_file_name)
{
    NVIC_SystemReset();
}


/**@brief Assert macro callback function.
 *
 * @details This function will be called in case of an assert in the
SoftDevice.
 *
 * @warning This handler is an example only and does not fit a final
product. You need to analyze
 *          how your product is supposed to react in case of Assert.
 * @warning On assert from the SoftDevice, the system can only recover on
reset.
```

```
 *
 * @param[in]   line_num   Line number of the failing ASSERT call.
 * @param[in]   file_name  File name of the failing ASSERT call.
 */
void assert_nrf_callback(uint16_t line_num, const uint8_t * p_file_name)
{
    app_error_handler(DEAD_BEEF, line_num, p_file_name);
}


/**@brief LEDs initialization.
 *
 * @details Initializes all LEDs used by the application.
 */
static void leds_init(void)
{
      GPIO_LED_CONFIG(OUTPUT1);
        GPIO_LED_CONFIG(OUTPUT2);
        nrf_gpio_pin_clear(OUTPUT1);
        nrf_gpio_pin_clear(OUTPUT2);
}

/*****************************************************************************
**
* Static Timeout Handling Functions
*****************************************************************************
*/

/**@brief Battery measurement timer timeout handler.
*
* @details This function will be called each time the battery level
measurement timer expires.
* This function will start the ADC.
*
* @param[in] p_context Pointer used for passing some arbitrary information
(context) from the
* app_start_timer() call to the timeout handler.
*/
static void time_adc_level_meas_timeout_handler(void * p_context)
{
    UNUSED_PARAMETER(p_context);
    battery_start();
}

/**@brief advertiser timer timeout handler.
 *
 * @details This function will be called each time advertiser timer expires.
 *
 * @param[in]   p_context   Pointer used for passing some arbitrary
information (context) from the
 *                          app_start_timer() call to the timeout handler.
 */
static void time_advertiser_timeout_handler(void * p_context)
{
    uint16_t err_code;


    if(stateOfAdvertise==MODE_ADV_ADVERTISE){
        err_code = sd_ble_gap_adv_stop();   //stop advertise
    APP_ERROR_CHECK(err_code);
```



```
        stateOfAdvertise=MODE_ADV_WAIT;     //change state
        err_code = app_timer_stop(m_advertiser_timer_id);
        APP_ERROR_CHECK(err_code);
        err_code = app_timer_start(m_advertiser_timer_id,

ADVERTISER_TIMER_INTERVAL_LENGTH,
                                                              NULL);
        APP_ERROR_CHECK(err_code);
    }
    else
            advertising_start();                    //start advertise
    //start timer advertise
            err_code = app_timer_stop(m_advertiser_timer_id);
            err_code = app_timer_start(m_advertiser_timer_id,

ADVERTISER_TIMER_INTERVAL_SHORT,
                                                              NULL);
            APP_ERROR_CHECK(err_code);
}



/**@brief shut down Output timer timeout handler.
 *
 * @details This function will be called each time solar LED timer expires.
 *
 * @param[in]   p_context   Pointer used for passing some arbitrary
information (context) from the
 *                          app_start_timer() call to the timeout handler.
 */
static void time_outPut_off_timeout_handler(void * p_context)
{
    UNUSED_PARAMETER(p_context);
        if (m_mms.mode==MODE_AUTOMATIQUE)
        {
            m_mms.outPut=OUTPUT_OFF;
            changeOutPut();
        }
}

/**@brief Timer initialization.
 *
 * @details Initializes the timer module.
 */
static void timers_init(void)
{
    // Initialize timer module, making it use the scheduler
    APP_TIMER_INIT(APP_TIMER_PRESCALER, APP_TIMER_MAX_TIMERS,
APP_TIMER_OP_QUEUE_SIZE, false);

    //Create timers
    uint32_t err_code = app_timer_create(&m_outPut_off_timer_id,
                                APP_TIMER_MODE_REPEATED,
                                time_outPut_off_timeout_handler);
    APP_ERROR_CHECK(err_code);

        //Create timers
    err_code = app_timer_create(&m_adc_timer_id,

                                APP_TIMER_MODE_REPEATED,
```

```
                                time_adc_level_meas_timeout_handler);
    APP_ERROR_CHECK(err_code);

            //Create timers
    err_code = app_timer_create(&m_advertiser_timer_id,
                                APP_TIMER_MODE_REPEATED,
                                time_advertiser_timeout_handler);
    APP_ERROR_CHECK(err_code);


}


/**@brief GAP initialization.
 *
 * @details This function shall be used to setup all the necessary GAP
(Generic Access Profile)
 *          parameters of the device. It also sets the permissions and
appearance.
 */
static void gap_params_init(void)
{
    uint32_t                err_code;
    ble_gap_conn_params_t   gap_conn_params;
    ble_gap_conn_sec_mode_t sec_mode;

    BLE_GAP_CONN_SEC_MODE_SET_OPEN(&sec_mode);

    err_code = sd_ble_gap_device_name_set(&sec_mode, DEVICE_NAME,
strlen(DEVICE_NAME));
    APP_ERROR_CHECK(err_code);


    err_code = sd_ble_gap_appearance_set(BLE_APPEARANCE_GENERIC_TAG);
    APP_ERROR_CHECK(err_code);

    memset(&gap_conn_params, 0, sizeof(gap_conn_params));

    gap_conn_params.min_conn_interval = MIN_CONN_INTERVAL;
    gap_conn_params.max_conn_interval = MAX_CONN_INTERVAL;
    gap_conn_params.slave_latency     = SLAVE_LATENCY;
    gap_conn_params.conn_sup_timeout  = CONN_SUP_TIMEOUT;

    err_code = sd_ble_gap_ppcp_set(&gap_conn_params);
    APP_ERROR_CHECK(err_code);
}


/**@brief Advertising functionality initialization.
 *
 * @details Encodes the required advertising data and passes it to the
stack.
 *          Also builds a structure to be passed to the stack when starting
advertising.
 */
/**@brief Advertising functionality initialization.
 *
 * @details Encodes the required advertising data and passes it to the
stack.
 *          Also builds a structure to be passed to the stack when starting
```



```
advertising.
 */
static void advertising_init(void)
{
    uint32_t      err_code;
    ble_advdata_t advdata;
    uint8_t       flags = BLE_GAP_ADV_FLAGS_LE_ONLY_GENERAL_DISC_MODE;
        stateOfAdvertise=MODE_ADV_WAIT;
    ble_uuid_t adv_uuids[] =
    {
        {BLE_UUID_HEART_RATE_SERVICE,         BLE_UUID_TYPE_BLE},
        {BLE_UUID_BATTERY_SERVICE,            BLE_UUID_TYPE_BLE},
        {BLE_UUID_DEVICE_INFORMATION_SERVICE, BLE_UUID_TYPE_BLE}
    };

    // Build and set advertising data
    memset(&advdata, 0, sizeof(advdata));

    advdata.name_type               = BLE_ADVDATA_FULL_NAME;
    advdata.include_appearance      = true;
    advdata.flags.size              = sizeof(flags);
    advdata.flags.p_data            = &flags;
    advdata.uuids_complete.uuid_cnt = sizeof(adv_uuids) /
sizeof(adv_uuids[0]);
    advdata.uuids_complete.p_uuids  = adv_uuids;

    err_code = ble_advdata_set(&advdata, NULL);
    APP_ERROR_CHECK(err_code);
}



/**@brief Initialize Radio Notification event handler.
 */
static void radio_notification_init(void)
{
    uint32_t err_code;

    err_code = ble_radio_notification_init(NRF_APP_PRIORITY_HIGH,

NRF_RADIO_NOTIFICATION_DISTANCE_4560US,
                                           ble_flash_on_radio_active_evt);
    APP_ERROR_CHECK(err_code);
}
/**@brief Initialize mms Service.
 */
static void mms_init(void)
{
    uint32_t       err_code;
    ble_mms_init_t mms_init_obj;
        m_mms.mode = MODE_AUTOMATIQUE;
    memset(&mms_init_obj, 0, sizeof(mms_init_obj));
    mms_init_obj.evt_handler = on_mms_evt;

    err_code = ble_mms_init(&m_mms , &mms_init_obj);
    APP_ERROR_CHECK(err_code);
}

/**@brief Initialize Battery Service.
 */
```

```
static void bas_init(void)
{
    uint32_t       err_code;
    ble_bas_init_t bas_init_obj;

    memset(&bas_init_obj, 0, sizeof(bas_init_obj));

    bas_init_obj.evt_handler          = NULL;
    bas_init_obj.support_notification = true;
    bas_init_obj.p_report_ref         = NULL;
    bas_init_obj.initial_batt_level   = 100;


BLE_GAP_CONN_SEC_MODE_SET_OPEN(&bas_init_obj.battery_level_char_attr_md.cccd
_write_perm);

BLE_GAP_CONN_SEC_MODE_SET_OPEN(&bas_init_obj.battery_level_char_attr_md.read
_perm);

BLE_GAP_CONN_SEC_MODE_SET_NO_ACCESS(&bas_init_obj.battery_level_char_attr_md
.write_perm);


BLE_GAP_CONN_SEC_MODE_SET_OPEN(&bas_init_obj.battery_level_report_read_perm)
;

    err_code = ble_bas_init(&m_bas, &bas_init_obj);
    APP_ERROR_CHECK(err_code);
}

/**@brief Initialize Battery Service.
 */
static void css_init(void)
{
    uint32_t       err_code;
    ble_css_init_t css_init_obj;

    memset(&css_init_obj, 0, sizeof(css_init_obj));

    css_init_obj.evt_handler          = NULL;
    css_init_obj.support_notification = true;
    css_init_obj.p_report_ref         = NULL;
    css_init_obj.initial_cels_level   = 100;


BLE_GAP_CONN_SEC_MODE_SET_OPEN(&css_init_obj.cellule_solar_level_char_attr_m
d.cccd_write_perm);

BLE_GAP_CONN_SEC_MODE_SET_OPEN(&css_init_obj.cellule_solar_level_char_attr_m
d.read_perm);

BLE_GAP_CONN_SEC_MODE_SET_NO_ACCESS(&css_init_obj.cellule_solar_level_char_a
ttr_md.write_perm);


BLE_GAP_CONN_SEC_MODE_SET_OPEN(&css_init_obj.cellule_solar_level_report_read
_perm);

    err_code = ble_css_init(&m_css, &css_init_obj);
    APP_ERROR_CHECK(err_code);
```



```
}

/**@brief Initialize services that will be used by the application.
 */
static void services_init(void)
{

    //initialize mms Service
        mms_init();

    //initialize batterie level Service
        bas_init();

    //initialize cellule solar level Service
        css_init();
}


/**@brief ADC interrupt handler.
* @details This function will fetch the conversion result from the ADC,
convert the value into
* percentage and send it to peer.
*/
void ADC_IRQHandler(void)
{
    if (NRF_ADC->EVENTS_END != 0)
    {
            //conversion ad for the battery lvl
            if(typeOfMeasAdc==ADC_MEAS_TYPE_BATTERY)
            {
                    uint8_t adc_result;
                    uint32_t err_code;

                    NRF_ADC->EVENTS_END = 0;
                    adc_result = NRF_ADC->RESULT;
                    NRF_ADC->TASKS_STOP = 1;


adc_result=(uint8_t)(((float)(adc_result))*((float)(FACTOR_BATTERY_LVL_IN_MI
LLIVOLT)));

                    err_code =
ble_bas_battery_level_update(&m_bas,adc_result ); //percentage_batt_lvl

                    if (
                            (err_code != NRF_SUCCESS)
                            &&
                            (err_code != NRF_ERROR_INVALID_STATE)
                            &&
                            (err_code != BLE_ERROR_NO_TX_BUFFERS)
                            &&
                            (err_code != BLE_ERROR_GATTS_SYS_ATTR_MISSING)
                    )
                    {
                            APP_ERROR_HANDLER(err_code);
                    }
                    cellule_solar_start();
            }
            //conversion ad for the cellule solar lvl
            else if(typeOfMeasAdc==ADC_MEAS_TYPE_CELLULE_SOLAR)
```

```
            {
                    uint8_t adc_result;
                    uint32_t err_code;

                    NRF_ADC->EVENTS_END = 0;
                    adc_result = NRF_ADC->RESULT;
                    NRF_ADC->TASKS_STOP = 1;


adc_result=(uint8_t)(((float)(adc_result))*((float)(FACTOR_CELL_SOL_LVL_IN_M
ILLIVOLT)));

                    err_code =
ble_css_cellule_solar_level_update(&m_css,adc_result );
//percentage_cels_lvl
                    if (
                            (err_code != NRF_SUCCESS)
                            &&
                            (err_code != NRF_ERROR_INVALID_STATE)
                            &&
                            (err_code != BLE_ERROR_NO_TX_BUFFERS)
                            &&
                            (err_code != BLE_ERROR_GATTS_SYS_ATTR_MISSING)
                    )
                    {
                            APP_ERROR_HANDLER(err_code);
                    }









            }
    }
}


void battery_start(void)
{
    uint32_t err_code;
        typeOfMeasAdc=ADC_MEAS_TYPE_BATTERY;

    // Configure ADC
    NRF_ADC->INTENSET = ADC_INTENSET_END_Msk;
    NRF_ADC->CONFIG = (ADC_CONFIG_RES_8bit << ADC_CONFIG_RES_Pos) |
                          (ADC_CONFIG_INPSEL_AnalogInputNoPrescaling  <<
ADC_CONFIG_INPSEL_Pos) |
                          (ADC_CONFIG_REFSEL_VBG << ADC_CONFIG_REFSEL_Pos) |
                          (INPUT_BATTERY_MEAS << ADC_CONFIG_PSEL_Pos) |
                          (ADC_CONFIG_EXTREFSEL_None <<
ADC_CONFIG_EXTREFSEL_Pos);
    NRF_ADC->EVENTS_END = 0;
    NRF_ADC->ENABLE = ADC_ENABLE_ENABLE_Enabled;

    // Enable ADC interrupt
    err_code = sd_nvic_ClearPendingIRQ(ADC_IRQn);
    APP_ERROR_CHECK(err_code);
```



```
    err_code = sd_nvic_SetPriority(ADC_IRQn, NRF_APP_PRIORITY_LOW);
    APP_ERROR_CHECK(err_code);

    err_code = sd_nvic_EnableIRQ(ADC_IRQn);
    APP_ERROR_CHECK(err_code);

    NRF_ADC->EVENTS_END = 0; // Stop any running conversions.
    NRF_ADC->TASKS_START = 1;
}

void cellule_solar_start(void)
{
    uint32_t err_code;
        typeOfMeasAdc=ADC_MEAS_TYPE_CELLULE_SOLAR;
    // Configure ADC
    NRF_ADC->INTENSET = ADC_INTENSET_END_Msk;
    NRF_ADC->CONFIG = (ADC_CONFIG_RES_8bit << ADC_CONFIG_RES_Pos) |
                          (ADC_CONFIG_INPSEL_AnalogInputNoPrescaling  <<
ADC_CONFIG_INPSEL_Pos) |
                          (ADC_CONFIG_REFSEL_VBG << ADC_CONFIG_REFSEL_Pos) |
                          (INPUT_CELL_SOL_MEAS << ADC_CONFIG_PSEL_Pos) |
                          (ADC_CONFIG_EXTREFSEL_None <<
ADC_CONFIG_EXTREFSEL_Pos);
    NRF_ADC->EVENTS_END = 0;
    NRF_ADC->ENABLE = ADC_ENABLE_ENABLE_Enabled;

    // Enable ADC interrupt
    err_code = sd_nvic_ClearPendingIRQ(ADC_IRQn);
    APP_ERROR_CHECK(err_code);

    err_code = sd_nvic_SetPriority(ADC_IRQn, NRF_APP_PRIORITY_LOW);
    APP_ERROR_CHECK(err_code);

    err_code = sd_nvic_EnableIRQ(ADC_IRQn);
    APP_ERROR_CHECK(err_code);

    NRF_ADC->EVENTS_END = 0; // Stop any running conversions.
    NRF_ADC->TASKS_START = 1;
}

/**@brief Initialize security parameters.
 */
static void sec_params_init(void)
{
    m_sec_params.timeout      = SEC_PARAM_TIMEOUT;
    m_sec_params.bond         = SEC_PARAM_BOND;
    m_sec_params.mitm         = SEC_PARAM_MITM;
    m_sec_params.io_caps      = SEC_PARAM_IO_CAPABILITIES;
    m_sec_params.oob          = SEC_PARAM_OOB;
    m_sec_params.min_key_size = SEC_PARAM_MIN_KEY_SIZE;
    m_sec_params.max_key_size = SEC_PARAM_MAX_KEY_SIZE;
}

/**@brief Connection Parameters Module handler.
 *
 * @details This function will be called for all events in the Connection
Parameters Module which
 *          are passed to the application.
 *          @note All this function does is to disconnect. This could have
been done by simply
```

```
 *                setting the disconnect_on_fail config parameter, but
instead we use the event
 *                handler mechanism to demonstrate its use.
 *
 * @param[in]   p_evt   Event received from the Connection Parameters
Module.
 */
static void on_conn_params_evt(ble_conn_params_evt_t * p_evt)
{
    uint32_t err_code;

    APP_ERROR_CHECK_BOOL(p_evt->evt_type == BLE_CONN_PARAMS_EVT_FAILED);

    err_code = sd_ble_gap_disconnect(m_conn_handle,
BLE_HCI_CONN_INTERVAL_UNACCEPTABLE);
    APP_ERROR_CHECK(err_code);
}


/**@brief Connection Parameters module error handler.
 *
 * @param[in]   nrf_error   Error code containing information about what
went wrong.
 */
static void conn_params_error_handler(uint32_t nrf_error)
{
    APP_ERROR_HANDLER(nrf_error);
}


/**@brief Initialize the Connection Parameters module.
 */
static void conn_params_init(void)
{
    uint32_t               err_code;
    ble_conn_params_init_t cp_init;

    memset(&cp_init, 0, sizeof(cp_init));

    cp_init.p_conn_params                  = NULL;
    cp_init.first_conn_params_update_delay = FIRST_CONN_PARAMS_UPDATE_DELAY;
    cp_init.next_conn_params_update_delay  = NEXT_CONN_PARAMS_UPDATE_DELAY;
    cp_init.max_conn_params_update_count   = MAX_CONN_PARAMS_UPDATE_COUNT;
    cp_init.start_on_notify_cccd_handle    = BLE_GATT_HANDLE_INVALID;
    cp_init.disconnect_on_fail             = false;
    cp_init.evt_handler                    = on_conn_params_evt;
    cp_init.error_handler                  = conn_params_error_handler;

    err_code = ble_conn_params_init(&cp_init);
    APP_ERROR_CHECK(err_code);
}



/**@brief Start advertising.
 */
static void advertising_start(void)
{
    uint32_t             err_code;
    ble_gap_adv_params_t adv_params;
```



```
        stateOfAdvertise=MODE_ADV_ADVERTISE;
    // Start advertising
    memset(&adv_params, 0, sizeof(adv_params));

    adv_params.type        = BLE_GAP_ADV_TYPE_ADV_IND;
    adv_params.p_peer_addr = NULL;
    adv_params.fp          = BLE_GAP_ADV_FP_ANY;
    adv_params.interval    = APP_ADV_INTERVAL;
    adv_params.timeout     = APP_ADV_TIMEOUT_IN_SECONDS;

    err_code = sd_ble_gap_adv_start(&adv_params);
    APP_ERROR_CHECK(err_code);

}



/**@brief Immediate Alert event handler.
 *
 * @details This function will be called for all Immediate Alert events
which are passed to the
 *          application.
 *
 * @param[in]   p_ias  Immediate Alert stucture.
 * @param[in]   p_evt  Event received from the Immediate Alert service.
 */
static void on_mms_evt(ble_mms_t * p_mms, ble_mms_evt_t * p_evt)
{
    switch (p_evt->evt_type)
    {
        case BLE_MMS_EVT_OUTPUT_UPDATED:
            //alert_signal(p_evt->params.alert_level);
            break;

        default:
            break;
    }
}



/**@brief Application's BLE Stack event handler.
 *
 * @param[in]   p_ble_evt   Bluetooth stack event.
 */
static void on_ble_evt(ble_evt_t * p_ble_evt)
{
    uint32_t                         err_code = NRF_SUCCESS;
    static ble_gap_evt_auth_status_t m_auth_status;
    ble_gap_enc_info_t *             p_enc_info;

    switch (p_ble_evt->header.evt_id)
    {
        case BLE_GAP_EVT_CONNECTED:
                        //make first mesure ad
                        battery_start();

                        //stop timer for advertise
                        err_code = app_timer_stop(m_advertiser_timer_id);
```

```
                        APP_ERROR_CHECK(err_code);

                // Start timer for battery lvl
                        err_code = app_timer_start(m_adc_timer_id,
ADCCONVERT_LEVEL_MEAS_INTERVAL, NULL);
                        APP_ERROR_CHECK(err_code);

                        //stop timer for shut down outPut
                        err_code = app_timer_stop(m_outPut_off_timer_id);
                        APP_ERROR_CHECK(err_code);


             break;

        case BLE_GAP_EVT_DISCONNECTED:
           if (m_mms.mode==MODE_AUTOMATIQUE)
                                    {
                                        //change output
                                        m_mms.outPut=OUTPUT1_ON;
                                        changeOutPut();
                                        // Start detection timer.
                                        err_code =
app_timer_start(m_outPut_off_timer_id,

OUTPUT_OFF_TIMER_INTERVAL,

NULL);
                                        APP_ERROR_CHECK(err_code);
                                    }
                                    //change mode on automatique of manuel
finish
                        else if((m_mms.mode
==MODE_MANUEL)&&(m_mms.outPut==0x00))
                                    {
                                        m_mms.mode=MODE_AUTOMATIQUE;
                                    }
                        m_conn_handle = BLE_CONN_HANDLE_INVALID;

                        if (err_code == NRF_SUCCESS)
                            {
                                //stop timer ad conversion
                                err_code = app_timer_stop(m_adc_timer_id);
                                APP_ERROR_CHECK(err_code);

                                //start timer advertise
                                stateOfAdvertise=MODE_ADV_WAIT;
                                err_code =
app_timer_start(m_advertiser_timer_id,

ADVERTISER_TIMER_INTERVAL_SHORT,

NULL);
                                APP_ERROR_CHECK(err_code);



                            }
                        break;
```



```
        case BLE_GAP_EVT_SEC_PARAMS_REQUEST:
            err_code = sd_ble_gap_sec_params_reply(m_conn_handle,

BLE_GAP_SEC_STATUS_SUCCESS,
                                                   &m_sec_params);
            break;

        case BLE_GATTS_EVT_SYS_ATTR_MISSING:
            err_code = sd_ble_gatts_sys_attr_set(m_conn_handle, NULL, 0);
            break;

        case BLE_GAP_EVT_AUTH_STATUS:
            m_auth_status = p_ble_evt->evt.gap_evt.params.auth_status;
            break;

        case BLE_GAP_EVT_SEC_INFO_REQUEST:
            p_enc_info = &m_auth_status.periph_keys.enc_info;
            if (p_enc_info->div ==
p_ble_evt->evt.gap_evt.params.sec_info_request.div)
            {
                err_code = sd_ble_gap_sec_info_reply(m_conn_handle,
p_enc_info, NULL);
            }
            else
            {
                // No keys found for this device
                err_code = sd_ble_gap_sec_info_reply(m_conn_handle, NULL,
NULL);
            }
            break;

        case BLE_GAP_EVT_TIMEOUT:

            break;

        default:
            break;
    }
    APP_ERROR_CHECK(err_code);
}



/**@brief Dispatches a BLE stack event to all modules with a BLE stack event
handler.
 *
 * @details This function is called from the scheduler in the main loop
after a BLE stack
 *          event has been received.
 *
 * @param[in]   p_ble_evt   Bluetooth stack event.
 */
static void ble_evt_dispatch(ble_evt_t * p_ble_evt)
{
        on_ble_evt(p_ble_evt);
    ble_conn_params_on_ble_evt(p_ble_evt);
        ble_mms_on_ble_evt(&m_mms, p_ble_evt);
        ble_bas_on_ble_evt(&m_bas, p_ble_evt);
        ble_css_on_ble_evt(&m_css, p_ble_evt);
```

```
}


/**@brief BLE stack initialization.
 *
 * @details Initializes the SoftDevice and the BLE event interrupt.
 */
static void ble_stack_init(void)
{
    BLE_STACK_HANDLER_INIT(  NRF_CLOCK_LFCLKSRC_SYNTH_250_PPM,
                             BLE_L2CAP_MTU_DEF,
                             ble_evt_dispatch,
                             false);

}
/**@brief Event Scheduler initialization.
 */
/*static void scheduler_init(void)
{
    APP_SCHED_INIT(SCHED_MAX_EVENT_DATA_SIZE, SCHED_QUEUE_SIZE);
}*/

/**@brief Initialize GPIOTE handler module.
 */
static void gpiote_init(void)
{
    APP_GPIOTE_INIT(APP_GPIOTE_MAX_USERS);
}



/**@brief Power manager.
 */
static void power_manage(void)
{
    uint32_t err_code = sd_app_event_wait();
    APP_ERROR_CHECK(err_code);
}

static void changeOutPut(void)
{
        switch (m_mms.outPut){
                    case OUTPUT1_ON:    nrf_gpio_pin_set(OUTPUT1);

nrf_gpio_pin_clear(OUTPUT2);
                                                    break;
                    case OUTPUT2_ON:    nrf_gpio_pin_clear(OUTPUT1);

nrf_gpio_pin_set(OUTPUT2);
                                                    break;
                    case OUTPUT12_ON:   nrf_gpio_pin_set(OUTPUT1);

nrf_gpio_pin_set(OUTPUT2);
                                                    break;
                    case OUTPUT_OFF:    nrf_gpio_pin_clear(OUTPUT1);

nrf_gpio_pin_clear(OUTPUT2);
                                                    break;
                    default:                    nrf_gpio_pin_clear(OUTPUT1);
```



```
nrf_gpio_pin_clear(OUTPUT2);
                                                                    break;
            }
}



/**@brief Application main function.
 */
int main(void)
{
    // Initialize

    leds_init();
    gpiote_init();
    ble_stack_init();
        ble_error_log_init();
      timers_init();
      //scheduler_init();
    gap_params_init();
    advertising_init();
    services_init();
    conn_params_init();
    sec_params_init();
        radio_notification_init();
    sd_power_mode_set(NRF_POWER_MODE_LOWPWR);

        m_mms.outPut=OUTPUT_OFF;
        m_mms.mode=MODE_AUTOMATIQUE;
    //start timer advertise
        uint16_t err_code;
      err_code = sd_power_mode_set(NRF_POWER_MODE_LOWPWR);
        APP_ERROR_CHECK(err_code);
        err_code = app_timer_start(m_advertiser_timer_id,

ADVERTISER_TIMER_INTERVAL_LENGTH,
                                                                    NULL);
        APP_ERROR_CHECK(err_code);

    // Enter main loop
    for (;;)
    {
            //app_sched_execute();
            changeOutPut();
      power_manage();
            //nrf_gpio_pin_toggle(OUTPUT1);
        }
}

/**
 * @}
 */
```

```
/* Copyright (c) 2012 Nordic Semiconductor. All Rights Reserved.
 *
 * The information contained herein is property of Nordic Semiconductor ASA.
 * Terms and conditions of usage are described in detail in NORDIC
 * SEMICONDUCTOR STANDARD SOFTWARE LICENSE AGREEMENT.
 *
 * Licensees are granted free, non-transferable use of the information. NO
 * WARRANTY of ANY KIND is provided. This heading must NOT be removed from
 * the file.
 *
 */

/** @file
 *
 * @defgroup ble_sdk_app_template_main main.c
 * @{
 * @ingroup ble_sdk_app_template
 * @brief  main file.
 *
 *
 * @author Jimmy Dubuis
 */


#include <stdint.h>
#include <string.h>

#include "nrf.h"
#include "app_error.h"
#include "nrf_gpio.h"
#include "nrf51_bitfields.h"
#include "boards/pca10001.h"
#include "ble.h"
#include "ble_hci.h"
#include "ble_srv_common.h"
#include "ble_advdata.h"
#include "ble_conn_params.h"
#include "ble_stack_handler.h"
#include "ble_radio_notification.h"
#include "ble_flash.h"
#include "ble_debug_assert_handler.h"
#include "ble_error_log.h"

#include "app_timer.h"
#include "app_gpiote.h"
#include "app_scheduler.h"

#include "nordic_common.h"
#include "ble_mms.h"
#include "ble_bas.h"
#include "ble_css.h"

#define DEVICE_NAME                     "Pres_VLAMPE"
/**< Name of device. Will be included in the advertising data. */
#define MANUFACTURER_NAME               "HES-SO Valais"
/**< Manufacturer. Will be passed to Device Information Service. */

#define APP_ADV_TIMEOUT_IN_SECONDS
BLE_GAP_ADV_TIMEOUT_GENERAL_UNLIMITED       /**< The advertising timeout (in
units of seconds). */
```



```
#define APP_ADV_INTERVAL                32
/**< The advertising interval (in units of 0.625 ms. This value corresponds
to 1 s). */
#define NBR_OF_ADVERTISE_IN_BURST               5
/**< number of advertise in burst */
#define BASE_TIME_INTERVAL                          625
/**< base time of the advertiser */

#define SECOND_1_25_MS_UNITS            800
/**< Definition of 1 second, when 1 unit is 1.25 ms. */
#define SECOND_10_MS_UNITS              100
/**< Definition of 1 second, when 1 unit is 10 ms. */
#define MIN_CONN_INTERVAL               (SECOND_1_25_MS_UNITS / 2)
/**< Minimum acceptable connection interval (0.5 seconds), Connection
interval uses 1.25 ms units. */
#define MAX_CONN_INTERVAL               (SECOND_1_25_MS_UNITS)
/**< Maximum acceptable connection interval (1 second), Connection interval
uses 1.25 ms units. */
#define SLAVE_LATENCY                   0
/**< Slave latency. */
#define CONN_SUP_TIMEOUT                (4 * SECOND_10_MS_UNITS)
/**< Connection supervisory timeout (4 seconds), Supervision Timeout uses 10
ms units. */

#define FIRST_CONN_PARAMS_UPDATE_DELAY  APP_TIMER_TICKS(5000,
APP_TIMER_PRESCALER) /**< Time from initiating event (connect or start of
notification) to first time sd_ble_gap_conn_param_update is called (15
seconds). */
#define NEXT_CONN_PARAMS_UPDATE_DELAY   APP_TIMER_TICKS(5000,
APP_TIMER_PRESCALER)  /**< Time between each call to
sd_ble_gap_conn_param_update after the first (5 seconds). */
#define MAX_CONN_PARAMS_UPDATE_COUNT    3
/**< Number of attempts before giving up the connection parameter
negotiation. */

#define APP_GPIOTE_MAX_USERS            1
/**< Maximum number of users of the GPIOTE handler. */

#define SEC_PARAM_TIMEOUT               30
/**< Timeout for Pairing Request or Security Request (in seconds). */
#define SEC_PARAM_BOND                  1
/**< Perform bonding. */
#define SEC_PARAM_MITM                  0
/**< Man In The Middle protection not required. */
#define SEC_PARAM_IO_CAPABILITIES       BLE_GAP_IO_CAPS_NONE
/**< No I/O capabilities. */
#define SEC_PARAM_OOB                   0
/**< Out Of Band data not available. */
#define SEC_PARAM_MIN_KEY_SIZE          7
/**< Minimum encryption key size. */
#define SEC_PARAM_MAX_KEY_SIZE          16
/**< Maximum encryption key size. */

#define DEAD_BEEF                       0xDEADBEEF
/**< Value used as error code on stack dump, can be used to identify stack
location on stack unwind. */
```

```
#define APP_TIMER_PRESCALER             0
/**< Value of the RTC1 PRESCALER register. */
#define APP_TIMER_MAX_TIMERS            4
/**< Maximum number of simultaneously created timers. */
#define APP_TIMER_OP_QUEUE_SIZE         5
/**< Size of timer operation queues. */


#define ADCCONVERT_LEVEL_MEAS_INTERVAL          APP_TIMER_TICKS(30000,
APP_TIMER_PRESCALER) /**< Battery level measurement interval (ticks). */
#define ADVERTISER_TIMER_INTERVAL_LENGTH    APP_TIMER_TICKS(2000,
APP_TIMER_PRESCALER) /**< timer for send the burst of advertise interval
(ticks). */
#define ADVERTISER_TIMER_INTERVAL_SHORT
APP_TIMER_TICKS(NBR_OF_ADVERTISE_IN_BURST*APP_ADV_INTERVAL*BASE_TIME_INTERVA
L/1000, APP_TIMER_PRESCALER) /**< timer for send the burst of advertise
interval (ticks). */
#define OUTPUT_OFF_TIMER_INTERVAL                   APP_TIMER_TICKS(4000,
APP_TIMER_PRESCALER) /**< timer for solar LED interval (ticks). */


#define SCHED_MAX_EVENT_DATA_SIZE       sizeof(app_timer_event_t)
/**< Maximum size of scheduler events. Note that scheduler BLE stack events
do not contain any data, as the events are being pulled from the stack in
the event handler. */
#define SCHED_QUEUE_SIZE                10
/**< Maximum number of events in the scheduler queue. */

#define MODE_ADV_WAIT                                   0
/**< value when adv is WAIT */
#define MODE_ADV_ADVERTISE                              1
/**< value when adv is ADVERTISER */

#define OUTPUT1                                             LED0
/**< pin of the LED0 */
#define OUTPUT2                                             LED1
/**< pin of the LED1*/
#define CMD_MEAS_BAT                                        1
/**< pin of the CMD for the battery */
#define CMD_MEAS_CEL                                        0
/**< pin of the CMD for the cel solar */


#define ADC_MEAS_TYPE_BATTERY               0
/**< value when the adc make an meas of the battery*/
#define ADC_MEAS_TYPE_CELLULE_SOLAR         1
/**< value when the adc make an meas of the cel solar*/


#define INPUT_CELL_SOL_MEAS     ADC_CONFIG_PSEL_AnalogInput0
/**< value of the analog input 0*/
#define INPUT_BATTERY_MEAS      ADC_CONFIG_PSEL_AnalogInput1
/**< value of the analog input 1*/

#define FACTOR_CELL_SOL_LVL_IN_MILLIVOLT    1.2/(254*0.4)*100
/**< factor for the conversion in millivolt for the cel solar*/
#define FACTOR_BATTERY_LVL_IN_MILLIVOLT     1.2/(254*0.4)*100
/**< factor for the conversion in millivoltfor the battery*/
```



```
static  uint16_t typeOfMeasAdc;  //define if the next adc is for the battery
or the cellule solar
static  uint16_t stateOfAdvertise;



static  app_timer_id_t                  m_outPut_off_timer_id;
/**< timer for shutdown the output before no connection */
static  app_timer_id_t                                  m_adc_timer_id;
/**< Battery timer. */
static  app_timer_id_t
m_advertiser_timer_id;                                              /**<
advertiser timer*/

static ble_gap_sec_params_t             m_sec_params;
/**< Security requirements for this application. */
static uint16_t                         m_conn_handle =
BLE_CONN_HANDLE_INVALID;    /**< Handle of the current connection. */

static ble_mms_t                       m_mms;
/**< Structure used to identify the Immediate Alert service. */
static ble_bas_t                        m_bas;
/**< Structure used to identify the battery service. */
static ble_css_t                        m_css;
/**< Structure used to identify the cellule solar service. */

static void battery_start(void);
static void cellule_solar_start(void);
static void changeOutPut(void);
static void on_mms_evt(ble_mms_t * p_mms, ble_mms_evt_t * p_evt);
static void advertising_start(void);



typedef enum
{
  TIMER0 = 0,  /*!< Timer0 module, base address at 0x40008000 */
  TIMER1,      /*!< Timer1 module, base address at 0x40009000 */
  TIMER2       /*!< Timer2 module, base address at 0x4000A000 */
} timer_t;

/**@brief Error handler function, which is called when an error has
occurred.
 *
 * @warning This handler is an example only and does not fit a final
product. You need to analyze
 *          how your product is supposed to react in case of error.
 *
 * @param[in] error_code  Error code supplied to the handler.
 * @param[in] line_num    Line number where the handler is called.
 * @param[in] p_file_name Pointer to the file name.
 */
void app_error_handler(uint32_t error_code, uint32_t line_num, const uint8_t
* p_file_name)
{
    NVIC_SystemReset();
```

```
}


/**@brief Assert macro callback function.
 *
 * @details This function will be called in case of an assert in the
SoftDevice.
 *
 * @warning This handler is an example only and does not fit a final
product. You need to analyze
 *          how your product is supposed to react in case of Assert.
 * @warning On assert from the SoftDevice, the system can only recover on
reset.
 *
 * @param[in]   line_num   Line number of the failing ASSERT call.
 * @param[in]   file_name  File name of the failing ASSERT call.
 */
void assert_nrf_callback(uint16_t line_num, const uint8_t * p_file_name)
{
    app_error_handler(DEAD_BEEF, line_num, p_file_name);
}


/**@brief LEDs initialization.
 *
 * @details Initializes all LEDs used by the application.
 */
static void leds_init(void)
{
      GPIO_LED_CONFIG(OUTPUT1);
        GPIO_LED_CONFIG(OUTPUT2);
        GPIO_LED_CONFIG(CMD_MEAS_BAT);
        GPIO_LED_CONFIG(CMD_MEAS_CEL);
        nrf_gpio_pin_clear(CMD_MEAS_BAT);
        nrf_gpio_pin_clear(CMD_MEAS_CEL);
        nrf_gpio_pin_clear(OUTPUT1);
        nrf_gpio_pin_clear(OUTPUT2);
}

/******************************************************************************
**
* Static Timeout Handling Functions
******************************************************************************
*/

/**@brief Battery measurement timer timeout handler.
*
* @details This function will be called each time the battery level
measurement timer expires.
* This function will start the ADC.
*
* @param[in] p_context Pointer used for passing some arbitrary information
(context) from the
* app_start_timer() call to the timeout handler.
*/
static void time_adc_level_meas_timeout_handler(void * p_context)
{
    UNUSED_PARAMETER(p_context);
    battery_start();
}
```



```

/**@brief advertiser timer timeout handler.
 *
 * @details This function will be called each time advertiser timer expires.
 *
 * @param[in]   p_context   Pointer used for passing some arbitrary
information (context) from the
 *                          app_start_timer() call to the timeout handler.
 */
static void time_advertiser_timeout_handler(void * p_context)
{
    uint16_t err_code;


    if(stateOfAdvertise==MODE_ADV_ADVERTISE){
        err_code = sd_ble_gap_adv_stop();   //stop advertise
    APP_ERROR_CHECK(err_code);
        stateOfAdvertise=MODE_ADV_WAIT;     //change state
        err_code = app_timer_stop(m_advertiser_timer_id);
        APP_ERROR_CHECK(err_code);
        err_code = app_timer_start(m_advertiser_timer_id,

ADVERTISER_TIMER_INTERVAL_LENGTH,
                                                                NULL);
        APP_ERROR_CHECK(err_code);
    }
    else
            advertising_start();                    //start advertise
    //start timer advertise
            err_code = app_timer_stop(m_advertiser_timer_id);
            err_code = app_timer_start(m_advertiser_timer_id,

ADVERTISER_TIMER_INTERVAL_SHORT,
                                                                NULL);
            APP_ERROR_CHECK(err_code);
}



/**@brief shut down Output timer timeout handler.
 *
 * @details This function will be called each time solar LED timer expires.
 *
 * @param[in]   p_context   Pointer used for passing some arbitrary
information (context) from the
 *                          app_start_timer() call to the timeout handler.
 */
static void time_outPut_off_timeout_handler(void * p_context)
{
    UNUSED_PARAMETER(p_context);
        if (m_mms.mode==MODE_AUTOMATIQUE)
        {
            m_mms.outPut=OUTPUT_OFF;
            changeOutPut();
        }
}

/**@brief Timer initialization.
 *
 * @details Initializes the timer module.
```

```
 * @return
 */
static void timers_init(void)
{
    // Initialize timer module, making it use the scheduler
    APP_TIMER_INIT(APP_TIMER_PRESCALER, APP_TIMER_MAX_TIMERS,
APP_TIMER_OP_QUEUE_SIZE, false);

    //Create timers
    uint32_t err_code = app_timer_create(&m_outPut_off_timer_id,
                                APP_TIMER_MODE_SINGLE_SHOT,
                                time_outPut_off_timeout_handler);
    APP_ERROR_CHECK(err_code);

        //Create timers
    err_code = app_timer_create(&m_adc_timer_id,
                                APP_TIMER_MODE_REPEATED,
                                time_adc_level_meas_timeout_handler);
    APP_ERROR_CHECK(err_code);

            //Create timers
    err_code = app_timer_create(&m_advertiser_timer_id,
                                APP_TIMER_MODE_SINGLE_SHOT,
                                time_advertiser_timeout_handler);
    APP_ERROR_CHECK(err_code);


}


/**@brief GAP initialization.
 *
 * @details This function shall be used to setup all the necessary GAP
(Generic Access Profile)
 *          parameters of the device. It also sets the permissions and
appearance.
 */
static void gap_params_init(void)
{
    uint32_t                err_code;
    ble_gap_conn_params_t   gap_conn_params;
    ble_gap_conn_sec_mode_t sec_mode;

    BLE_GAP_CONN_SEC_MODE_SET_OPEN(&sec_mode);

    err_code = sd_ble_gap_device_name_set(&sec_mode, DEVICE_NAME,
strlen(DEVICE_NAME));
    APP_ERROR_CHECK(err_code);


    err_code = sd_ble_gap_appearance_set(BLE_APPEARANCE_GENERIC_TAG);
    APP_ERROR_CHECK(err_code);

    memset(&gap_conn_params, 0, sizeof(gap_conn_params));

    gap_conn_params.min_conn_interval = MIN_CONN_INTERVAL;
    gap_conn_params.max_conn_interval = MAX_CONN_INTERVAL;
    gap_conn_params.slave_latency     = SLAVE_LATENCY;
    gap_conn_params.conn_sup_timeout  = CONN_SUP_TIMEOUT;
```



```
    err_code = sd_ble_gap_ppcp_set(&gap_conn_params);
    APP_ERROR_CHECK(err_code);
}


/**@brief Advertising functionality initialization.
 *
 * @details Encodes the required advertising data and passes it to the
stack.
 *          Also builds a structure to be passed to the stack when starting
advertising.
 */
/**@brief Advertising functionality initialization.
 *
 * @details Encodes the required advertising data and passes it to the
stack.
 *          Also builds a structure to be passed to the stack when starting
advertising.
 */
static void advertising_init(void)
{
    uint32_t      err_code;
    ble_advdata_t advdata;
    uint8_t       flags = BLE_GAP_ADV_FLAGS_LE_ONLY_GENERAL_DISC_MODE;
        stateOfAdvertise=MODE_ADV_WAIT;
    ble_uuid_t adv_uuids[] =
    {
        {BLE_UUID_HEART_RATE_SERVICE,         BLE_UUID_TYPE_BLE},
        {BLE_UUID_BATTERY_SERVICE,            BLE_UUID_TYPE_BLE},
        {BLE_UUID_DEVICE_INFORMATION_SERVICE, BLE_UUID_TYPE_BLE}
    };

    // Build and set advertising data
    memset(&advdata, 0, sizeof(advdata));

    advdata.name_type               = BLE_ADVDATA_FULL_NAME;
    advdata.include_appearance      = true;
    advdata.flags.size              = sizeof(flags);
    advdata.flags.p_data            = &flags;
    advdata.uuids_complete.uuid_cnt = sizeof(adv_uuids) /
sizeof(adv_uuids[0]);
    advdata.uuids_complete.p_uuids  = adv_uuids;

    err_code = ble_advdata_set(&advdata, NULL);
    APP_ERROR_CHECK(err_code);
}



/**@brief Initialize Radio Notification event handler.
 */
static void radio_notification_init(void)
{
    uint32_t err_code;

    err_code = ble_radio_notification_init(NRF_APP_PRIORITY_HIGH,

NRF_RADIO_NOTIFICATION_DISTANCE_4560US,
                                           ble_flash_on_radio_active_evt);
    APP_ERROR_CHECK(err_code);
```

```
}
/**@brief Initialize mms Service.
 */
static void mms_init(void)
{
    uint32_t       err_code;
    ble_mms_init_t mms_init_obj;
        m_mms.mode = MODE_AUTOMATIQUE;
    memset(&mms_init_obj, 0, sizeof(mms_init_obj));
    mms_init_obj.evt_handler = on_mms_evt;

    err_code = ble_mms_init(&m_mms , &mms_init_obj);
    APP_ERROR_CHECK(err_code);
}

/**@brief Initialize Battery Service.
 */
static void bas_init(void)
{
    uint32_t       err_code;
    ble_bas_init_t bas_init_obj;

    memset(&bas_init_obj, 0, sizeof(bas_init_obj));

    bas_init_obj.evt_handler          = NULL;
    bas_init_obj.support_notification = true;
    bas_init_obj.p_report_ref         = NULL;
    bas_init_obj.initial_batt_level   = 100;


BLE_GAP_CONN_SEC_MODE_SET_OPEN(&bas_init_obj.battery_level_char_attr_md.cccd
_write_perm);

BLE_GAP_CONN_SEC_MODE_SET_OPEN(&bas_init_obj.battery_level_char_attr_md.read
_perm);

BLE_GAP_CONN_SEC_MODE_SET_NO_ACCESS(&bas_init_obj.battery_level_char_attr_md
.write_perm);


BLE_GAP_CONN_SEC_MODE_SET_OPEN(&bas_init_obj.battery_level_report_read_perm)
;

    err_code = ble_bas_init(&m_bas, &bas_init_obj);
    APP_ERROR_CHECK(err_code);
}

/**@brief Initialize Battery Service.
 */
static void css_init(void)
{
    uint32_t       err_code;
    ble_css_init_t css_init_obj;

    memset(&css_init_obj, 0, sizeof(css_init_obj));

    css_init_obj.evt_handler          = NULL;
    css_init_obj.support_notification = true;
    css_init_obj.p_report_ref         = NULL;
    css_init_obj.initial_cels_level   = 100;
```



```


BLE_GAP_CONN_SEC_MODE_SET_OPEN(&css_init_obj.cellule_solar_level_char_attr_m
d.cccd_write_perm);

BLE_GAP_CONN_SEC_MODE_SET_OPEN(&css_init_obj.cellule_solar_level_char_attr_m
d.read_perm);

BLE_GAP_CONN_SEC_MODE_SET_NO_ACCESS(&css_init_obj.cellule_solar_level_char_a
ttr_md.write_perm);


BLE_GAP_CONN_SEC_MODE_SET_OPEN(&css_init_obj.cellule_solar_level_report_read
_perm);

    err_code = ble_css_init(&m_css, &css_init_obj);
    APP_ERROR_CHECK(err_code);
}

/**@brief Initialize services that will be used by the application.
 */
static void services_init(void)
{

    //initialize mms Service
        mms_init();

    //initialize batterie level Service
        bas_init();

    //initialize cellule solar level Service
        css_init();
}


/**@brief ADC interrupt handler.
* @details This function will fetch the conversion result from the ADC,
convert the value into
* percentage and send it to peer.
*/
void ADC_IRQHandler(void)
{
    if (NRF_ADC->EVENTS_END != 0)
    {
            //conversion ad for the battery lvl
            if(typeOfMeasAdc==ADC_MEAS_TYPE_BATTERY)
            {
                    uint8_t adc_result;
                    uint32_t err_code;

                    NRF_ADC->EVENTS_END = 0;
                    adc_result = NRF_ADC->RESULT;
                    NRF_ADC->TASKS_STOP = 1;


adc_result=(uint8_t)(((float)(adc_result))*((float)(FACTOR_BATTERY_LVL_IN_MI
LLIVOLT)));
                    nrf_gpio_pin_clear(CMD_MEAS_BAT);
                    err_code =
ble_bas_battery_level_update(&m_bas,adc_result ); //percentage_batt_lvl
```

```
                    if (
                            (err_code != NRF_SUCCESS)
                            &&
                            (err_code != NRF_ERROR_INVALID_STATE)
                            &&
                            (err_code != BLE_ERROR_NO_TX_BUFFERS)
                            &&
                            (err_code != BLE_ERROR_GATTS_SYS_ATTR_MISSING)
                    )
                    {
                            APP_ERROR_HANDLER(err_code);
                    }
                    cellule_solar_start();
            }
            //conversion ad for the cellule solar lvl
            else if(typeOfMeasAdc==ADC_MEAS_TYPE_CELLULE_SOLAR)
            {
                    uint8_t adc_result;
                    uint32_t err_code;

                    NRF_ADC->EVENTS_END = 0;
                    adc_result = NRF_ADC->RESULT;
                    NRF_ADC->TASKS_STOP = 1;


adc_result=(uint8_t)(((float)(adc_result))*((float)(FACTOR_CELL_SOL_LVL_IN_M
ILLIVOLT)));

                    err_code =
ble_css_cellule_solar_level_update(&m_css,adc_result );
//percentage_cels_lvl
                    nrf_gpio_pin_clear(CMD_MEAS_CEL);
                    if (
                            (err_code != NRF_SUCCESS)
                            &&
                            (err_code != NRF_ERROR_INVALID_STATE)
                            &&
                            (err_code != BLE_ERROR_NO_TX_BUFFERS)
                            &&
                            (err_code != BLE_ERROR_GATTS_SYS_ATTR_MISSING)
                    )
                    {
                            APP_ERROR_HANDLER(err_code);
                    }









            }
    }
}

/**@brief start the conversion ad for the battery lvl
 *
 */
void battery_start(void)
```



```
{
    uint32_t err_code;
        typeOfMeasAdc=ADC_MEAS_TYPE_BATTERY;
        nrf_gpio_pin_set(CMD_MEAS_BAT);
    // Configure ADC
    NRF_ADC->INTENSET = ADC_INTENSET_END_Msk;
    NRF_ADC->CONFIG = (ADC_CONFIG_RES_8bit << ADC_CONFIG_RES_Pos) |
                          (ADC_CONFIG_INPSEL_AnalogInputNoPrescaling  <<
ADC_CONFIG_INPSEL_Pos) |
                          (ADC_CONFIG_REFSEL_VBG << ADC_CONFIG_REFSEL_Pos) |
                          (INPUT_BATTERY_MEAS << ADC_CONFIG_PSEL_Pos) |
                          (ADC_CONFIG_EXTREFSEL_None <<
ADC_CONFIG_EXTREFSEL_Pos);
    NRF_ADC->EVENTS_END = 0;
    NRF_ADC->ENABLE = ADC_ENABLE_ENABLE_Enabled;

    // Enable ADC interrupt
    err_code = sd_nvic_ClearPendingIRQ(ADC_IRQn);
    APP_ERROR_CHECK(err_code);

    err_code = sd_nvic_SetPriority(ADC_IRQn, NRF_APP_PRIORITY_LOW);
    APP_ERROR_CHECK(err_code);

    err_code = sd_nvic_EnableIRQ(ADC_IRQn);
    APP_ERROR_CHECK(err_code);

    NRF_ADC->EVENTS_END = 0; // Stop any running conversions.
    NRF_ADC->TASKS_START = 1;
}
/**@brief start the conversion ad for the solar cell lvl
 *
 */
void cellule_solar_start(void)
{
    uint32_t err_code;
        typeOfMeasAdc=ADC_MEAS_TYPE_CELLULE_SOLAR;
        nrf_gpio_pin_set(CMD_MEAS_CEL);
    // Configure ADC
    NRF_ADC->INTENSET = ADC_INTENSET_END_Msk;
    NRF_ADC->CONFIG = (ADC_CONFIG_RES_8bit << ADC_CONFIG_RES_Pos) |
                          (ADC_CONFIG_INPSEL_AnalogInputNoPrescaling  <<
ADC_CONFIG_INPSEL_Pos) |
                          (ADC_CONFIG_REFSEL_VBG << ADC_CONFIG_REFSEL_Pos) |
                          (INPUT_CELL_SOL_MEAS << ADC_CONFIG_PSEL_Pos) |
                          (ADC_CONFIG_EXTREFSEL_None <<
ADC_CONFIG_EXTREFSEL_Pos);
    NRF_ADC->EVENTS_END = 0;
    NRF_ADC->ENABLE = ADC_ENABLE_ENABLE_Enabled;

    // Enable ADC interrupt
    err_code = sd_nvic_ClearPendingIRQ(ADC_IRQn);
    APP_ERROR_CHECK(err_code);

    err_code = sd_nvic_SetPriority(ADC_IRQn, NRF_APP_PRIORITY_LOW);
    APP_ERROR_CHECK(err_code);

    err_code = sd_nvic_EnableIRQ(ADC_IRQn);
    APP_ERROR_CHECK(err_code);

    NRF_ADC->EVENTS_END = 0; // Stop any running conversions.
```

```
    NRF_ADC->TASKS_START = 1;
}

/**@brief Initialize security parameters.
 */
static void sec_params_init(void)
{
    m_sec_params.timeout      = SEC_PARAM_TIMEOUT;
    m_sec_params.bond         = SEC_PARAM_BOND;
    m_sec_params.mitm         = SEC_PARAM_MITM;
    m_sec_params.io_caps      = SEC_PARAM_IO_CAPABILITIES;
    m_sec_params.oob          = SEC_PARAM_OOB;
    m_sec_params.min_key_size = SEC_PARAM_MIN_KEY_SIZE;
    m_sec_params.max_key_size = SEC_PARAM_MAX_KEY_SIZE;
}

/**@brief Connection Parameters Module handler.
 *
 * @details This function will be called for all events in the Connection
Parameters Module which
 *          are passed to the application.
 *          @note All this function does is to disconnect. This could have
been done by simply
 *                setting the disconnect_on_fail config parameter, but
instead we use the event
 *                handler mechanism to demonstrate its use.
 *
 * @param[in]   p_evt   Event received from the Connection Parameters
Module.
 */
static void on_conn_params_evt(ble_conn_params_evt_t * p_evt)
{
    uint32_t err_code;

    APP_ERROR_CHECK_BOOL(p_evt->evt_type == BLE_CONN_PARAMS_EVT_FAILED);

    err_code = sd_ble_gap_disconnect(m_conn_handle,
BLE_HCI_CONN_INTERVAL_UNACCEPTABLE);
    APP_ERROR_CHECK(err_code);
}


/**@brief Connection Parameters module error handler.
 *
 * @param[in]   nrf_error   Error code containing information about what
went wrong.
 */
static void conn_params_error_handler(uint32_t nrf_error)
{
    APP_ERROR_HANDLER(nrf_error);
}


/**@brief Initialize the Connection Parameters module.
 */
static void conn_params_init(void)
{
    uint32_t               err_code;
    ble_conn_params_init_t cp_init;
```



```
    memset(&cp_init, 0, sizeof(cp_init));

    cp_init.p_conn_params                  = NULL;
    cp_init.first_conn_params_update_delay = FIRST_CONN_PARAMS_UPDATE_DELAY;
    cp_init.next_conn_params_update_delay  = NEXT_CONN_PARAMS_UPDATE_DELAY;
    cp_init.max_conn_params_update_count   = MAX_CONN_PARAMS_UPDATE_COUNT;
    cp_init.start_on_notify_cccd_handle    = BLE_GATT_HANDLE_INVALID;
    cp_init.disconnect_on_fail             = false;
    cp_init.evt_handler                    = on_conn_params_evt;
    cp_init.error_handler                  = conn_params_error_handler;

    err_code = ble_conn_params_init(&cp_init);
    APP_ERROR_CHECK(err_code);
}


/**@brief Start advertising.
 */
static void advertising_start(void)
{
    uint32_t             err_code;
    ble_gap_adv_params_t adv_params;

        stateOfAdvertise=MODE_ADV_ADVERTISE;
    // Start advertising
    memset(&adv_params, 0, sizeof(adv_params));

    adv_params.type        = BLE_GAP_ADV_TYPE_ADV_IND;
    adv_params.p_peer_addr = NULL;
    adv_params.fp          = BLE_GAP_ADV_FP_ANY;
    adv_params.interval    = APP_ADV_INTERVAL;
    adv_params.timeout     = APP_ADV_TIMEOUT_IN_SECONDS;

    err_code = sd_ble_gap_adv_start(&adv_params);
    APP_ERROR_CHECK(err_code);

}


/**@brief mode manual event handler.
 *
 * @details This function will be called for all mode manual events which
are passed to the
 *          application.
 *
 * @param[in]   p_mms  mode manual stucture.
 * @param[in]   p_evt  Event received from mode manual service.
 */
static void on_mms_evt(ble_mms_t * p_mms, ble_mms_evt_t * p_evt)
{
    switch (p_evt->evt_type)
    {
        case BLE_MMS_EVT_OUTPUT_UPDATED:
            break;

        default:
            break;
    }
```

```
}



/**@brief Application's BLE Stack event handler.
 *
 * @param[in]   p_ble_evt   Bluetooth stack event.
 */
static void on_ble_evt(ble_evt_t * p_ble_evt)
{
    uint32_t                         err_code = NRF_SUCCESS;
    static ble_gap_evt_auth_status_t m_auth_status;
    ble_gap_enc_info_t *             p_enc_info;

    switch (p_ble_evt->header.evt_id)
    {
        case BLE_GAP_EVT_CONNECTED:
                         //make first mesure ad
                         battery_start();

                         //stop timer for advertise
                         err_code = app_timer_stop(m_advertiser_timer_id);
                         APP_ERROR_CHECK(err_code);

                // Start timer for battery lvl
                         err_code = app_timer_start(m_adc_timer_id,
ADCCONVERT_LEVEL_MEAS_INTERVAL, NULL);
                         APP_ERROR_CHECK(err_code);

                         //stop timer for shut down outPut
                         err_code = app_timer_stop(m_outPut_off_timer_id);
                         APP_ERROR_CHECK(err_code);


            break;

        case BLE_GAP_EVT_DISCONNECTED:
           if (m_mms.mode==MODE_AUTOMATIQUE)
                                    {
                                        //change output
                                        m_mms.outPut=OUTPUT1_ON;
                                        changeOutPut();
                                        // Start detection timer.
                                        err_code =
app_timer_start(m_outPut_off_timer_id,

OUTPUT_OFF_TIMER_INTERVAL,

NULL);
                                        APP_ERROR_CHECK(err_code);
                                    }
                                    //change mode on automatique of manuel
finish
                         else if((m_mms.mode
==MODE_MANUEL)&&(m_mms.outPut==0x00))
                                    {
                                        m_mms.mode=MODE_AUTOMATIQUE;
                                    }
                         m_conn_handle = BLE_CONN_HANDLE_INVALID;
```



```
                         if (err_code == NRF_SUCCESS)
                             {
                                 //stop timer ad conversion
                                 err_code = app_timer_stop(m_adc_timer_id);
                                 APP_ERROR_CHECK(err_code);

                                 //start timer advertise
                                 stateOfAdvertise=MODE_ADV_WAIT;
                                 err_code =
app_timer_start(m_advertiser_timer_id,

ADVERTISER_TIMER_INTERVAL_SHORT,

NULL);
                                 APP_ERROR_CHECK(err_code);




                             }
                         break;

        case BLE_GAP_EVT_SEC_PARAMS_REQUEST:
            err_code = sd_ble_gap_sec_params_reply(m_conn_handle,

BLE_GAP_SEC_STATUS_SUCCESS,
                                                   &m_sec_params);
            break;

        case BLE_GATTS_EVT_SYS_ATTR_MISSING:
            err_code = sd_ble_gatts_sys_attr_set(m_conn_handle, NULL, 0);
            break;

        case BLE_GAP_EVT_AUTH_STATUS:
            m_auth_status = p_ble_evt->evt.gap_evt.params.auth_status;
            break;

        case BLE_GAP_EVT_SEC_INFO_REQUEST:
            p_enc_info = &m_auth_status.periph_keys.enc_info;
            if (p_enc_info->div ==
p_ble_evt->evt.gap_evt.params.sec_info_request.div)
            {
                err_code = sd_ble_gap_sec_info_reply(m_conn_handle,
p_enc_info, NULL);
            }
            else
            {
                // No keys found for this device
                err_code = sd_ble_gap_sec_info_reply(m_conn_handle, NULL,
NULL);
            }
            break;

        case BLE_GAP_EVT_TIMEOUT:

            break;

        default:
            break;
    }
```

```
    APP_ERROR_CHECK(err_code);
}



/**@brief Dispatches a BLE stack event to all modules with a BLE stack event handler.
 *
 * @details This function is called from the scheduler in the main loop after a BLE stack
 *          event has been received.
 *
 * @param[in]   p_ble_evt   Bluetooth stack event.
 */
static void ble_evt_dispatch(ble_evt_t * p_ble_evt)
{
        on_ble_evt(p_ble_evt);
    ble_conn_params_on_ble_evt(p_ble_evt);
        ble_mms_on_ble_evt(&m_mms, p_ble_evt);
        ble_bas_on_ble_evt(&m_bas, p_ble_evt);
        ble_css_on_ble_evt(&m_css, p_ble_evt);

}


/**@brief BLE stack initialization.
 *
 * @details Initializes the SoftDevice and the BLE event interrupt.
 */
static void ble_stack_init(void)
{
    BLE_STACK_HANDLER_INIT(  NRF_CLOCK_LFCLKSRC_SYNTH_250_PPM,
                             BLE_L2CAP_MTU_DEF,
                             ble_evt_dispatch,
                             true);

}
/**@brief Event Scheduler initialization.
 */
static void scheduler_init(void)
{
    APP_SCHED_INIT(SCHED_MAX_EVENT_DATA_SIZE, SCHED_QUEUE_SIZE);
}

/**@brief Initialize GPIOTE handler module.
 */
static void gpiote_init(void)
{
    APP_GPIOTE_INIT(APP_GPIOTE_MAX_USERS);
}



/**@brief Power manager.
 */
static void power_manage(void)
{
    uint32_t err_code = sd_app_event_wait();
    APP_ERROR_CHECK(err_code);
```



```
}
/**@brief change the value of the output
 */
static void changeOutPut(void)
{
        switch (m_mms.outPut){
                    case OUTPUT1_ON:    nrf_gpio_pin_set(OUTPUT1);

nrf_gpio_pin_clear(OUTPUT2);
                                                            break;
                    case OUTPUT2_ON:    nrf_gpio_pin_clear(OUTPUT1);

nrf_gpio_pin_set(OUTPUT2);
                                                            break;
                    case OUTPUT12_ON:   nrf_gpio_pin_set(OUTPUT1);

nrf_gpio_pin_set(OUTPUT2);
                                                            break;
                    case OUTPUT_OFF:    nrf_gpio_pin_clear(OUTPUT1);

nrf_gpio_pin_clear(OUTPUT2);
                                                            break;
                    default:                    nrf_gpio_pin_clear(OUTPUT1);

nrf_gpio_pin_clear(OUTPUT2);
                                                            break;
           }
}



/**@brief Application main function.
 */
int main(void)
{
    // Initialize
    leds_init();
    gpiote_init();
    ble_stack_init();
        ble_error_log_init();
      timers_init();
      scheduler_init();
    gap_params_init();
    advertising_init();
    services_init();
    conn_params_init();
    sec_params_init();
        radio_notification_init();
    m_mms.outPut=OUTPUT_OFF;
        m_mms.mode=MODE_AUTOMATIQUE;
    //start timer advertise
        uint16_t err_code;
        err_code = app_timer_start(m_advertiser_timer_id,
                                   ADVERTISER_TIMER_INTERVAL_LENGTH,
                                   NULL);
        APP_ERROR_CHECK(err_code);

    // Enter main loop
    for (;;)
    {
```

```
    //app_sched_execute();
    changeOutPut();
    power_manage();
    }
}

/**
 * @}
 */
```

```
/* Copyright (c) 2012 Nordic Semiconductor. All Rights Reserved.
 *
 * The information contained herein is property of Nordic Semiconductor ASA.
 * Terms and conditions of usage are described in detail in NORDIC
 * SEMICONDUCTOR STANDARD SOFTWARE LICENSE AGREEMENT.
 *
 * Licensees are granted free, non-transferable use of the information. NO
 * WARRANTY of ANY KIND is provided. This heading must NOT be removed from
 * the file.
 *
 */

/** @file
 *
 * @defgroup ble_sdk_srv_css Cellule solaire Service
 * @{
 * @ingroup ble_sdk_srv
 * @brief Cellule solaire Service module.
 *
 * @details This module implements the Cellule solaire Service with the
Cellule solaire Level characteristic.
 *          During initialization it adds the Cellule solaire Service and
Cellule solaire Level characteristic
 *          to the BLE stack datacsse. Optionally it can also add a Report
Reference descriptor
 *          to the Cellule solaire Level characteristic (used when including
the Cellule solaire Service in
 *          the HID service).
 *
 *          If specified, the module will support notification of the
Cellule solaire Level characteristic
 *          through the ble_css_cellule_solar_level_update() function.
 *          If an event handler is supplied by the application, the Cellule
solaire Service will
 *          generate Cellule solaire Service events to the application.
 *
 * @note The application must propagate BLE stack events to the Cellule
solaire Service module by calling
 *       ble_css_on_ble_evt() from the from the @ref ble_stack_handler
callback.
 */

#ifndef BLE_CSS_H__
#define BLE_CSS_H__

#include <stdint.h>
#include <stdbool.h>
#include "ble.h"
#include "ble_srv_common.h"

/**@brief Cellule solaire Service event type. */
typedef enum
{
    BLE_CSS_EVT_NOTIFICATION_ENABLED,                             /**<
Cellule solaire value notification enabled event. */
    BLE_CSS_EVT_NOTIFICATION_DISABLED                             /**<
Cellule solaire value notification disabled event. */
} ble_css_evt_type_t;

/**@brief Cellule solaire Service event. */
```



```
typedef struct
{
    ble_css_evt_type_t evt_type;                                  /**< Type
of event. */
} ble_css_evt_t;

// Forward declaration of the ble_css_t type.
typedef struct ble_css_s ble_css_t;

/**@brief Cellule solaire Service event handler type. */
typedef void (*ble_css_evt_handler_t) (ble_css_t * p_css, ble_css_evt_t *
p_evt);

/**@brief Cellule solaire Service init structure. This contains all options
and data needed for
 *        initialization of the service.*/
typedef struct
{
    ble_css_evt_handler_t         evt_handler;                    /**< Event
handler to be called for handling events in the Cellule solaire Service. */
    bool                          support_notification;           /**< TRUE
if notification of Cellule solaire Level measurement is supported. */
    ble_srv_report_ref_t *        p_report_ref;                   /**< If
not NULL, a Report Reference descriptor with the specified value will be
added to the Cellule solaire Level characteristic */
    uint8_t                       initial_cels_level;             /**<
Initial cellule_solar level */
    ble_srv_cccd_security_mode_t  cellule_solar_level_char_attr_md;     /**<
Initial security level for cellule_solar characteristics attribute */
    ble_gap_conn_sec_mode_t       cellule_solar_level_report_read_perm; /**<
Initial security level for cellule_solar report read attribute */
} ble_css_init_t;

/**@brief Cellule solaire Service structure. This contains various status
information for the service. */
typedef struct ble_css_s
{
    ble_css_evt_handler_t         evt_handler;                    /**< Event
handler to be called for handling events in the Cellule solaire Service. */
    uint16_t                      service_handle;                 /**<
Handle of Cellule solaire Service (as provided by the BLE stack). */
    ble_gatts_char_handles_t      cellule_solar_level_handles;          /**<
Handles related to the Cellule solaire Level characteristic. */
    uint16_t                      report_ref_handle;              /**<
Handle of the Report Reference descriptor. */
    uint8_t                       cellule_solar_level_last;             /**<
Last Cellule solaire Level measurement passed to the Cellule solaire
Service. */
    uint16_t                      conn_handle;                    /**<
Handle of the current connection (as provided by the BLE stack, is
BLE_CONN_HANDLE_INVALID if not in a connection). */
    bool                          is_notification_supported;      /**< TRUE
if notification of Cellule solaire Level is supported. */
} ble_css_t;

/**@brief Initialize the Cellule solaire Service.
 *
 * @param[out]  p_css       Cellule solaire Service structure. This
structure will have to be supplied by
 *                          the application. It will be initialized by this
```

```
function, and will later
 *                           be used to identify this particular service
instance.
 * @param[in]   p_css_init  Information needed to initialize the service.
 *
 * @return      NRF_SUCCESS on successful initialization of service,
otherwise an error code.
 */
uint32_t ble_css_init(ble_css_t * p_css, const ble_css_init_t * p_css_init);

/**@brief Cellule solaire Service BLE stack event handler.
 *
 * @details Handles all events from the BLE stack of interest to the Cellule
solaire Service.
 *
 * @note For the requirements in the CSS specification to be fulfilled,
 *       ble_css_cellule_solar_level_update() must be called upon
reconnection if the
 *       cellule_solar level has changed while the service has been
disconnected from a bonded
 *       client.
 *
 * @param[in]   p_css      Cellule solaire Service structure.
 * @param[in]   p_ble_evt  Event received from the BLE stack.
 */
void ble_css_on_ble_evt(ble_css_t * p_css, ble_evt_t * p_ble_evt);

/**@brief Update cellule_solar level.
 *
 * @details The application calls this function after having performed a
cellule_solar measurement. If
 *          notification has been enabled, the cellule_solar level
characteristic is sent to the client.
 *
 * @note For the requirements in the CSS specification to be fulfilled,
 *       this function must be called upon reconnection if the cellule_solar
level has changed
 *       while the service has been disconnected from a bonded client.
 *
 * @param[in]   p_css          Cellule solaire Service structure.
 * @param[in]   cellule_solar_level  New cellule_solar measurement value (in
percent of full capacity).
 *
 * @return      NRF_SUCCESS on success, otherwise an error code.
 */
uint32_t ble_css_cellule_solar_level_update(ble_css_t * p_css, uint8_t
cellule_solar_level);

#endif // BLE_CSS_H__

/** @} */
```

```
/* Copyright (c) 2012 Nordic Semiconductor. All Rights Reserved.
 *
 * The information contained herein is property of Nordic Semiconductor ASA.
 * Terms and conditions of usage are described in detail in NORDIC
 * SEMICONDUCTOR STANDARD SOFTWARE LICENSE AGREEMENT.
 *
 * Licensees are granted free, non-transferable use of the information. NO
 * WARRANTY of ANY KIND is provided. This heading must NOT be removed from
 * the file.
 *
 */


#include <string.h>
#include "nordic_common.h"
#include "ble_srv_common.h"
#include "app_util.h"
#include "ble_css.h"

#define INVALID_CELLULE_SOLAR_LEVEL  255


/**@brief Connect event handler.
 *
 * @param[in]   p_css       cellule solar Service structure.
 * @param[in]   p_ble_evt   Event received from the BLE stack.
 */
static void on_connect(ble_css_t * p_css, ble_evt_t * p_ble_evt)
{
    p_css->conn_handle = p_ble_evt->evt.gap_evt.conn_handle;
}


/**@brief Disconnect event handler.
 *
 * @param[in]   p_css       cellule solar Service structure.
 * @param[in]   p_ble_evt   Event received from the BLE stack.
 */
static void on_disconnect(ble_css_t * p_css, ble_evt_t * p_ble_evt)
{
    UNUSED_PARAMETER(p_ble_evt);
    p_css->conn_handle = BLE_CONN_HANDLE_INVALID;
}


/**@brief Write event handler.
 *
 * @param[in]   p_css       cellule solar service structure.
 * @param[in]   p_ble_evt   Event received from the BLE stack.
 */
static void on_write(ble_css_t * p_css, ble_evt_t * p_ble_evt)
{
    if (p_css->is_notification_supported)
    {
        ble_gatts_evt_write_t * p_evt_write =
&p_ble_evt->evt.gatts_evt.params.write;

        if (
            (p_evt_write->handle ==
p_css->cellule_solar_level_handles.cccd_handle)
```



```
            &&
            (p_evt_write->len == 2)
        )
        {
            // CCCD written, call application event handler
            if (p_css->evt_handler != NULL)
            {
                ble_css_evt_t evt;

                if (ble_srv_is_notification_enabled(p_evt_write->data))
                {
                    evt.evt_type = BLE_CSS_EVT_NOTIFICATION_ENABLED;
                }
                else
                {
                    evt.evt_type = BLE_CSS_EVT_NOTIFICATION_DISABLED;
                }

                p_css->evt_handler(p_css, &evt);
            }
        }
    }
}


void ble_css_on_ble_evt(ble_css_t * p_css, ble_evt_t * p_ble_evt)
{
    switch (p_ble_evt->header.evt_id)
    {
        case BLE_GAP_EVT_CONNECTED:
            on_connect(p_css, p_ble_evt);
            break;

        case BLE_GAP_EVT_DISCONNECTED:
            on_disconnect(p_css, p_ble_evt);
            break;

        case BLE_GATTS_EVT_WRITE:
            on_write(p_css, p_ble_evt);
            break;

        default:
            break;
    }
}


/**@brief Add Battery Level characteristic.
 *
 * @param[in]   p_css        cellule solar Service structure.
 * @param[in]   p_css_init   Information needed to initialize the service.
 *
 * @return      NRF_SUCCESS on success, otherwise an error code.
 */
static uint32_t cellule_solar_level_char_add(ble_css_t * p_css, const
ble_css_init_t * p_css_init)
{
    uint32_t            err_code;
    ble_gatts_char_md_t char_md;
    ble_gatts_attr_md_t cccd_md;
```

```
    ble_gatts_attr_t    attr_char_value;
    ble_uuid_t          ble_uuid;
    ble_gatts_attr_md_t attr_md;
    uint8_t             initial_cellule_solar_level;
    uint8_t             encoded_report_ref[BLE_SRV_ENCODED_REPORT_REF_LEN];
    uint8_t             init_len;

    // Add cellule solar Level characteristic
    if (p_css->is_notification_supported)
    {
        memset(&cccd_md, 0, sizeof(cccd_md));

        // According to BAS_SPEC_V10, the read operation on cccd should be
possible without
        // authentication.
        BLE_GAP_CONN_SEC_MODE_SET_OPEN(&cccd_md.read_perm);
        cccd_md.write_perm =
p_css_init->cellule_solar_level_char_attr_md.cccd_write_perm;
        cccd_md.vloc = BLE_GATTS_VLOC_STACK;
    }

    memset(&char_md, 0, sizeof(char_md));

    char_md.char_props.read   = 1;
    char_md.char_props.notify = (p_css->is_notification_supported) ? 1 : 0;
    char_md.p_char_user_desc  = NULL;
    char_md.p_char_pf         = NULL;
    char_md.p_user_desc_md    = NULL;
    char_md.p_cccd_md         = (p_css->is_notification_supported) ?
&cccd_md : NULL;
    char_md.p_sccd_md         = NULL;

    BLE_UUID_BLE_ASSIGN(ble_uuid, BLE_UUID_DEVICE_INFORMATION_SERVICE);

    memset(&attr_md, 0, sizeof(attr_md));

    attr_md.read_perm  =
p_css_init->cellule_solar_level_char_attr_md.read_perm;
    attr_md.write_perm =
p_css_init->cellule_solar_level_char_attr_md.write_perm;
    attr_md.vloc       = BLE_GATTS_VLOC_STACK;
    attr_md.rd_auth    = 0;
    attr_md.wr_auth    = 0;
    attr_md.vlen       = 0;

    initial_cellule_solar_level = p_css_init->initial_cels_level;

    memset(&attr_char_value, 0, sizeof(attr_char_value));

    attr_char_value.p_uuid       = &ble_uuid;
    attr_char_value.p_attr_md    = &attr_md;
    attr_char_value.init_len     = sizeof(uint8_t);
    attr_char_value.init_offs    = 0;
    attr_char_value.max_len      = sizeof(uint8_t);
    attr_char_value.p_value      = &initial_cellule_solar_level;

    err_code = sd_ble_gatts_characteristic_add(p_css->service_handle,
&char_md,
                                               &attr_char_value,
```



```
&p_css->cellule_solar_level_handles);
    if (err_code != NRF_SUCCESS)
    {
        return err_code;
    }

    if (p_css_init->p_report_ref != NULL)
    {
        // Add Report Reference descriptor
        BLE_UUID_BLE_ASSIGN(ble_uuid, BLE_UUID_REPORT_REF_DESCR);

        memset(&attr_md, 0, sizeof(attr_md));

        attr_md.read_perm =
p_css_init->cellule_solar_level_report_read_perm;
        BLE_GAP_CONN_SEC_MODE_SET_NO_ACCESS(&attr_md.write_perm);

        attr_md.vloc       = BLE_GATTS_VLOC_STACK;
        attr_md.rd_auth    = 0;
        attr_md.wr_auth    = 0;
        attr_md.vlen       = 0;

        init_len = ble_srv_report_ref_encode(encoded_report_ref,
p_css_init->p_report_ref);

        memset(&attr_char_value, 0, sizeof(attr_char_value));

        attr_char_value.p_uuid       = &ble_uuid;
        attr_char_value.p_attr_md    = &attr_md;
        attr_char_value.init_len     = init_len;
        attr_char_value.init_offs    = 0;
        attr_char_value.max_len      = attr_char_value.init_len;
        attr_char_value.p_value      = encoded_report_ref;

        err_code =
sd_ble_gatts_descriptor_add(p_css->cellule_solar_level_handles.value_handle,
                                                &attr_char_value,
                                                &p_css->report_ref_handle);
        if (err_code != NRF_SUCCESS)
        {
            return err_code;
        }
    }
    else
    {
        p_css->report_ref_handle = BLE_GATT_HANDLE_INVALID;
    }

    return NRF_SUCCESS;
}


uint32_t ble_css_init(ble_css_t * p_css, const ble_css_init_t * p_css_init)
{
    uint32_t   err_code;
    ble_uuid_t ble_uuid;

    // Initialize service structure
    p_css->evt_handler               = p_css_init->evt_handler;
    p_css->conn_handle               = BLE_CONN_HANDLE_INVALID;
```

```
    p_css->is_notification_supported = p_css_init->support_notification;
    p_css->cellule_solar_level_last        = INVALID_CELLULE_SOLAR_LEVEL;

    // Add service
    BLE_UUID_BLE_ASSIGN(ble_uuid, BLE_UUID_DEVICE_INFORMATION_SERVICE);

    err_code = sd_ble_gatts_service_add(BLE_GATTS_SRVC_TYPE_PRIMARY,
&ble_uuid, &p_css->service_handle);
    if (err_code != NRF_SUCCESS)
    {
        return err_code;
    }

    // Add cellule_solar level characteristic
    return cellule_solar_level_char_add(p_css, p_css_init);
}


uint32_t ble_css_cellule_solar_level_update(ble_css_t * p_css, uint8_t
cellule_solar_level)
{
    uint32_t err_code = NRF_SUCCESS;

    if (cellule_solar_level != p_css->cellule_solar_level_last)
    {
        uint16_t len = sizeof(uint8_t);

        // Save new cellule_solar value
        p_css->cellule_solar_level_last = cellule_solar_level;

        // Update datacsse
        err_code =
sd_ble_gatts_value_set(p_css->cellule_solar_level_handles.value_handle,
                                       0,
                                       &len,
                                       &cellule_solar_level);
        if (err_code != NRF_SUCCESS)
        {
            return err_code;
        }

        // Send value if connected and notifying
        if ((p_css->conn_handle != BLE_CONN_HANDLE_INVALID) &&
p_css->is_notification_supported)
        {
            ble_gatts_hvx_params_t hvx_params;

            memset(&hvx_params, 0, sizeof(hvx_params));
            len = sizeof(uint8_t);

            hvx_params.handle   =
p_css->cellule_solar_level_handles.value_handle;
            hvx_params.type     = BLE_GATT_HVX_NOTIFICATION;
            hvx_params.offset   = 0;
            hvx_params.p_len    = &len;
            hvx_params.p_data   = &cellule_solar_level;

            err_code = sd_ble_gatts_hvx(p_css->conn_handle, &hvx_params);
        }
        else
```



```
        {
            err_code = NRF_ERROR_INVALID_STATE;
        }
    }

    return err_code;
}
```

```
/* Copyright (c) 2012 Nordic Semiconductor. All Rights Reserved.
 *
 * The information contained herein is property of Nordic Semiconductor ASA.
 * Terms and conditions of usage are described in detail in NORDIC
 * SEMICONDUCTOR STANDARD SOFTWARE LICENSE AGREEMENT.
 *
 * Licensees are granted free, non-transferable use of the information. NO
 * WARRANTY of ANY KIND is provided. This heading must NOT be removed from
 * the file.
 *
 */

/** @file
 *
 * @defgroup ble_sdk_srv_bas Battery Service
 * @{
 * @ingroup ble_sdk_srv
 * @brief Battery Service module.
 *
 * @details This module implements the Battery Service with the Battery
Level characteristic.
 *          During initialization it adds the Battery Service and Battery
Level characteristic
 *          to the BLE stack database. Optionally it can also add a Report
Reference descriptor
 *          to the Battery Level characteristic (used when including the
Battery Service in
 *          the HID service).
 *
 *          If specified, the module will support notification of the
Battery Level characteristic
 *          through the ble_bas_battery_level_update() function.
 *          If an event handler is supplied by the application, the Battery
Service will
 *          generate Battery Service events to the application.
 *
 * @note The application must propagate BLE stack events to the Battery
Service module by calling
 *       ble_bas_on_ble_evt() from the from the @ref ble_stack_handler
callback.
 */

#ifndef BLE_BAS_H__
#define BLE_BAS_H__

#include <stdint.h>
#include <stdbool.h>
#include "ble.h"
#include "ble_srv_common.h"

/**@brief Battery Service event type. */
typedef enum
{
    BLE_BAS_EVT_NOTIFICATION_ENABLED,                             /**<
Battery value notification enabled event. */
    BLE_BAS_EVT_NOTIFICATION_DISABLED                             /**<
Battery value notification disabled event. */
} ble_bas_evt_type_t;

/**@brief Battery Service event. */
```



```
typedef struct
{
    ble_bas_evt_type_t evt_type;                                  /**< Type
of event. */
} ble_bas_evt_t;

// Forward declaration of the ble_bas_t type.
typedef struct ble_bas_s ble_bas_t;

/**@brief Battery Service event handler type. */
typedef void (*ble_bas_evt_handler_t) (ble_bas_t * p_bas, ble_bas_evt_t *
p_evt);

/**@brief Battery Service init structure. This contains all options and data
needed for
 *        initialization of the service.*/
typedef struct
{
    ble_bas_evt_handler_t         evt_handler;                    /**< Event
handler to be called for handling events in the Battery Service. */
    bool                          support_notification;           /**< TRUE
if notification of Battery Level measurement is supported. */
    ble_srv_report_ref_t *        p_report_ref;                   /**< If
not NULL, a Report Reference descriptor with the specified value will be
added to the Battery Level characteristic */
    uint8_t                       initial_batt_level;             /**<
Initial battery level */
    ble_srv_cccd_security_mode_t  battery_level_char_attr_md;     /**<
Initial security level for battery characteristics attribute */
    ble_gap_conn_sec_mode_t       battery_level_report_read_perm; /**<
Initial security level for battery report read attribute */
} ble_bas_init_t;

/**@brief Battery Service structure. This contains various status
information for the service. */
typedef struct ble_bas_s
{
    ble_bas_evt_handler_t         evt_handler;                    /**< Event
handler to be called for handling events in the Battery Service. */
    uint16_t                      service_handle;                 /**<
Handle of Battery Service (as provided by the BLE stack). */
    ble_gatts_char_handles_t      battery_level_handles;          /**<
Handles related to the Battery Level characteristic. */
    uint16_t                      report_ref_handle;              /**<
Handle of the Report Reference descriptor. */
    uint8_t                       battery_level_last;             /**< Last
Battery Level measurement passed to the Battery Service. */
    uint16_t                      conn_handle;                    /**<
Handle of the current connection (as provided by the BLE stack, is
BLE_CONN_HANDLE_INVALID if not in a connection). */
    bool                          is_notification_supported;      /**< TRUE
if notification of Battery Level is supported. */
} ble_bas_t;

/**@brief Initialize the Battery Service.
 *
 * @param[out]  p_bas       Battery Service structure. This structure will
have to be supplied by
 *                          the application. It will be initialized by this
function, and will later
```

```
 *                          be used to identify this particular service
instance.
 * @param[in]   p_bas_init  Information needed to initialize the service.
 *
 * @return      NRF_SUCCESS on successful initialization of service,
otherwise an error code.
 */
uint32_t ble_bas_init(ble_bas_t * p_bas, const ble_bas_init_t * p_bas_init);

/**@brief Battery Service BLE stack event handler.
 *
 * @details Handles all events from the BLE stack of interest to the Battery
Service.
 *
 * @note For the requirements in the BAS specification to be fulfilled,
 *       ble_bas_battery_level_update() must be called upon reconnection if
the
 *       battery level has changed while the service has been disconnected
from a bonded
 *       client.
 *
 * @param[in]   p_bas      Battery Service structure.
 * @param[in]   p_ble_evt  Event received from the BLE stack.
 */
void ble_bas_on_ble_evt(ble_bas_t * p_bas, ble_evt_t * p_ble_evt);

/**@brief Update battery level.
 *
 * @details The application calls this function after having performed a
battery measurement. If
 *          notification has been enabled, the battery level characteristic
is sent to the client.
 *
 * @note For the requirements in the BAS specification to be fulfilled,
 *       this function must be called upon reconnection if the battery level
has changed
 *       while the service has been disconnected from a bonded client.
 *
 * @param[in]   p_bas          Battery Service structure.
 * @param[in]   battery_level  New battery measurement value (in percent of
full capacity).
 *
 * @return      NRF_SUCCESS on success, otherwise an error code.
 */
uint32_t ble_bas_battery_level_update(ble_bas_t * p_bas, uint8_t
battery_level);

#endif // BLE_BAS_H__

/** @} */
```

```
/* Copyright (c) 2012 Nordic Semiconductor. All Rights Reserved.
 *
 * The information contained herein is property of Nordic Semiconductor ASA.
 * Terms and conditions of usage are described in detail in NORDIC
 * SEMICONDUCTOR STANDARD SOFTWARE LICENSE AGREEMENT.
 *
 * Licensees are granted free, non-transferable use of the information. NO
 * WARRANTY of ANY KIND is provided. This heading must NOT be removed from
 * the file.
 *
 */

#include "ble_bas.h"
#include <string.h>
#include "nordic_common.h"
#include "ble_srv_common.h"
#include "app_util.h"


#define INVALID_BATTERY_LEVEL  255


/**@brief Connect event handler.
 *
 * @param[in]   p_bas       Battery Service structure.
 * @param[in]   p_ble_evt   Event received from the BLE stack.
 */
static void on_connect(ble_bas_t * p_bas, ble_evt_t * p_ble_evt)
{
    p_bas->conn_handle = p_ble_evt->evt.gap_evt.conn_handle;
}


/**@brief Disconnect event handler.
 *
 * @param[in]   p_bas       Battery Service structure.
 * @param[in]   p_ble_evt   Event received from the BLE stack.
 */
static void on_disconnect(ble_bas_t * p_bas, ble_evt_t * p_ble_evt)
{
    UNUSED_PARAMETER(p_ble_evt);
    p_bas->conn_handle = BLE_CONN_HANDLE_INVALID;
}


/**@brief Write event handler.
 *
 * @param[in]   p_bas       Battery Service structure.
 * @param[in]   p_ble_evt   Event received from the BLE stack.
 */
static void on_write(ble_bas_t * p_bas, ble_evt_t * p_ble_evt)
{
    if (p_bas->is_notification_supported)
    {
        ble_gatts_evt_write_t * p_evt_write =
&p_ble_evt->evt.gatts_evt.params.write;

        if (
            (p_evt_write->handle ==
p_bas->battery_level_handles.cccd_handle)
```



```
            &&
            (p_evt_write->len == 2)
        )
        {
            // CCCD written, call application event handler
            if (p_bas->evt_handler != NULL)
            {
                ble_bas_evt_t evt;

                if (ble_srv_is_notification_enabled(p_evt_write->data))
                {
                    evt.evt_type = BLE_BAS_EVT_NOTIFICATION_ENABLED;
                }
                else
                {
                    evt.evt_type = BLE_BAS_EVT_NOTIFICATION_DISABLED;
                }

                p_bas->evt_handler(p_bas, &evt);
            }
        }
    }
}


void ble_bas_on_ble_evt(ble_bas_t * p_bas, ble_evt_t * p_ble_evt)
{
    switch (p_ble_evt->header.evt_id)
    {
        case BLE_GAP_EVT_CONNECTED:
            on_connect(p_bas, p_ble_evt);
            break;

        case BLE_GAP_EVT_DISCONNECTED:
            on_disconnect(p_bas, p_ble_evt);
            break;

        case BLE_GATTS_EVT_WRITE:
            on_write(p_bas, p_ble_evt);
            break;

        default:
            break;
    }
}


/**@brief Add Battery Level characteristic.
 *
 * @param[in]   p_bas        Battery Service structure.
 * @param[in]   p_bas_init   Information needed to initialize the service.
 *
 * @return      NRF_SUCCESS on success, otherwise an error code.
 */
static uint32_t battery_level_char_add(ble_bas_t * p_bas, const
ble_bas_init_t * p_bas_init)
{
    uint32_t            err_code;
    ble_gatts_char_md_t char_md;
    ble_gatts_attr_md_t cccd_md;
```

```
    ble_gatts_attr_t    attr_char_value;
    ble_uuid_t          ble_uuid;
    ble_gatts_attr_md_t attr_md;
    uint8_t             initial_battery_level;
    uint8_t             encoded_report_ref[BLE_SRV_ENCODED_REPORT_REF_LEN];
    uint8_t             init_len;

    // Add Battery Level characteristic
    if (p_bas->is_notification_supported)
    {
        memset(&cccd_md, 0, sizeof(cccd_md));

        // According to BAS_SPEC_V10, the read operation on cccd should be
possible without
        // authentication.
        BLE_GAP_CONN_SEC_MODE_SET_OPEN(&cccd_md.read_perm);
        cccd_md.write_perm =
p_bas_init->battery_level_char_attr_md.cccd_write_perm;
        cccd_md.vloc = BLE_GATTS_VLOC_STACK;
    }

    memset(&char_md, 0, sizeof(char_md));

    char_md.char_props.read   = 1;
    char_md.char_props.notify = (p_bas->is_notification_supported) ? 1 : 0;
    char_md.p_char_user_desc  = NULL;
    char_md.p_char_pf         = NULL;
    char_md.p_user_desc_md    = NULL;
    char_md.p_cccd_md         = (p_bas->is_notification_supported) ?
&cccd_md : NULL;
    char_md.p_sccd_md         = NULL;

    BLE_UUID_BLE_ASSIGN(ble_uuid, BLE_UUID_BATTERY_LEVEL_CHAR);

    memset(&attr_md, 0, sizeof(attr_md));

    attr_md.read_perm  = p_bas_init->battery_level_char_attr_md.read_perm;
    attr_md.write_perm = p_bas_init->battery_level_char_attr_md.write_perm;
    attr_md.vloc       = BLE_GATTS_VLOC_STACK;
    attr_md.rd_auth    = 0;
    attr_md.wr_auth    = 0;
    attr_md.vlen       = 0;

    initial_battery_level = p_bas_init->initial_batt_level;

    memset(&attr_char_value, 0, sizeof(attr_char_value));

    attr_char_value.p_uuid       = &ble_uuid;
    attr_char_value.p_attr_md    = &attr_md;
    attr_char_value.init_len     = sizeof(uint8_t);
    attr_char_value.init_offs    = 0;
    attr_char_value.max_len      = sizeof(uint8_t);
    attr_char_value.p_value      = &initial_battery_level;

    err_code = sd_ble_gatts_characteristic_add(p_bas->service_handle,
&char_md,
                                               &attr_char_value,

&p_bas->battery_level_handles);
    if (err_code != NRF_SUCCESS)
```



```
    {
        return err_code;
    }

    if (p_bas_init->p_report_ref != NULL)
    {
        // Add Report Reference descriptor
        BLE_UUID_BLE_ASSIGN(ble_uuid, BLE_UUID_REPORT_REF_DESCR);

        memset(&attr_md, 0, sizeof(attr_md));

        attr_md.read_perm = p_bas_init->battery_level_report_read_perm;
        BLE_GAP_CONN_SEC_MODE_SET_NO_ACCESS(&attr_md.write_perm);

        attr_md.vloc       = BLE_GATTS_VLOC_STACK;
        attr_md.rd_auth    = 0;
        attr_md.wr_auth    = 0;
        attr_md.vlen       = 0;

        init_len = ble_srv_report_ref_encode(encoded_report_ref,
p_bas_init->p_report_ref);

        memset(&attr_char_value, 0, sizeof(attr_char_value));

        attr_char_value.p_uuid       = &ble_uuid;
        attr_char_value.p_attr_md    = &attr_md;
        attr_char_value.init_len     = init_len;
        attr_char_value.init_offs    = 0;
        attr_char_value.max_len      = attr_char_value.init_len;
        attr_char_value.p_value      = encoded_report_ref;

        err_code =
sd_ble_gatts_descriptor_add(p_bas->battery_level_handles.value_handle,
                                                &attr_char_value,
                                                &p_bas->report_ref_handle);
        if (err_code != NRF_SUCCESS)
        {
            return err_code;
        }
    }
    else
    {
        p_bas->report_ref_handle = BLE_GATT_HANDLE_INVALID;
    }

    return NRF_SUCCESS;
}


uint32_t ble_bas_init(ble_bas_t * p_bas, const ble_bas_init_t * p_bas_init)
{
    uint32_t   err_code;
    ble_uuid_t ble_uuid;

    // Initialize service structure
    p_bas->evt_handler               = p_bas_init->evt_handler;
    p_bas->conn_handle               = BLE_CONN_HANDLE_INVALID;
    p_bas->is_notification_supported = p_bas_init->support_notification;
    p_bas->battery_level_last        = INVALID_BATTERY_LEVEL;
```

```
    // Add service
    BLE_UUID_BLE_ASSIGN(ble_uuid, BLE_UUID_BATTERY_SERVICE);

    err_code = sd_ble_gatts_service_add(BLE_GATTS_SRVC_TYPE_PRIMARY,
&ble_uuid, &p_bas->service_handle);
    if (err_code != NRF_SUCCESS)
    {
        return err_code;
    }

    // Add battery level characteristic
    return battery_level_char_add(p_bas, p_bas_init);
}


uint32_t ble_bas_battery_level_update(ble_bas_t * p_bas, uint8_t
battery_level)
{
    uint32_t err_code = NRF_SUCCESS;

    if (battery_level != p_bas->battery_level_last)
    {
        uint16_t len = sizeof(uint8_t);

        // Save new battery value
        p_bas->battery_level_last = battery_level;

        // Update database
        err_code =
sd_ble_gatts_value_set(p_bas->battery_level_handles.value_handle,
                                       0,
                                       &len,
                                       &battery_level);
        if (err_code != NRF_SUCCESS)
        {
            return err_code;
        }

        // Send value if connected and notifying
        if ((p_bas->conn_handle != BLE_CONN_HANDLE_INVALID) &&
p_bas->is_notification_supported)
        {
            ble_gatts_hvx_params_t hvx_params;

            memset(&hvx_params, 0, sizeof(hvx_params));
            len = sizeof(uint8_t);

            hvx_params.handle   = p_bas->battery_level_handles.value_handle;
            hvx_params.type     = BLE_GATT_HVX_NOTIFICATION;
            hvx_params.offset   = 0;
            hvx_params.p_len    = &len;
            hvx_params.p_data   = &battery_level;

            err_code = sd_ble_gatts_hvx(p_bas->conn_handle, &hvx_params);
        }
        else
        {
            err_code = NRF_ERROR_INVALID_STATE;
        }
    }
```



```
    return err_code;
}
```

```
/* Copyright (c) 2012 Nordic Semiconductor. All Rights Reserved.
 *
 * The information contained herein is property of Nordic Semiconductor ASA.
 * Terms and conditions of usage are described in detail in NORDIC
 * SEMICONDUCTOR STANDARD SOFTWARE LICENSE AGREEMENT.
 *
 * Licensees are granted free, non-transferable use of the information. NO
 * WARRANTY of ANY KIND is provided. This heading must NOT be removed from
 * the file.
 *
 */

/** @file
 *
 * @defgroup ble_sdk_srv_ias mode manual Service
 * @{
 * @ingroup ble_sdk_srv
 * @brief mode manual Service module.
 *
 * @details This module implements the mode manual Service with the Alert
Level characteristic.
 *          During initialization it adds the mode manual Service and Alert
Level characteristic
 *          to the BLE stack database.
 *
 *          The application must supply an event handler for receiving mode
manual Service
 *          events. Using this handler, the service will notify the
application when the
 *          Alert Level characteristic value changes.
 *
 *          The service also provides a function for letting the application
poll the current
 *          value of the Alert Level characteristic.
 *
 * @note The application must propagate BLE stack events to the mode manual
Service
 *       module by calling ble_ias_on_ble_evt() from the @ref
ble_stack_handler callback.
 */

#ifndef BLE_MMS_H__
#define BLE_MMS_H__

#include <stdint.h>
#include "ble.h"

#define MODE_AUTOMATIQUE 0  /**< value when the mode is automatique */
#define MODE_MANUEL 1               /**< value when the mode is manual */

#define INITIAL_OUTPUT 0        /**< value initial for the output */
#define OUTPUT_OFF 0                /**< LED0&LED1 = 0*/
#define OUTPUT1_ON 1                /**< LED0=1 */
#define OUTPUT2_ON 2                /**< LED1=1 */
#define OUTPUT12_ON 3               /**< LED0 & LED1 =1 */

/**@brief mode manual Service event type. */
typedef enum
{
    BLE_MMS_EVT_OUTPUT_UPDATED                    /**< Alert Level Updated
```



```
event. */
} ble_mms_evt_type_t;

/**@brief mode manual Service event. */
typedef struct
{
    ble_mms_evt_type_t evt_type;                        /**< Type of event.
*/
    union
    {
        uint8_t outPut;                             /**< . */
    } params;
} ble_mms_evt_t;

// Forward declaration of the ble_ias_t type.
typedef struct ble_mms_s ble_mms_t;

/**@brief mode manual Service event handler type. */
typedef void (*ble_mms_evt_handler_t) (ble_mms_t * p_mms, ble_mms_evt_t *
p_evt);

/**@brief mode manual Service init structure. This contains all options and
data needed for
 *        initialization of the service. */
typedef struct
{
    ble_mms_evt_handler_t evt_handler;                  /**< Event handler
to be called for handling events in the mode manual Service. */
} ble_mms_init_t;

/**@brief mode manual Service structure. This contains various status
information for the
 *        service. */
typedef struct ble_mms_s
{
    ble_mms_evt_handler_t     evt_handler;              /**< Event handler
to be called for handling events in the mode manual Service. */
    uint16_t                  service_handle;           /**< Handle of mode
manual Service (as provided by the BLE stack). */
    ble_gatts_char_handles_t  outPut_handles;      /**< Handles related to
the Alert Level characteristic. */
        uint8_t mode;
        uint8_t outPut;
} ble_mms_t;

/**@brief Initialize the mode manual Service.
 *
 * @param[out]  p_mms       mode manual Service structure. This structure
will have to be
 *                          supplied by the application. It will be
initialized by this function,
 *                          and will later be used to identify this
particular service instance.
 * @param[in]   p_ias_init  Information needed to initialize the service.
 *
 * @return      NRF_SUCCESS on successful initialization of service,
otherwise an error code.
 */
uint32_t ble_mms_init(ble_mms_t * p_mms, const ble_mms_init_t * p_mms_init);
```

```
/**@brief mode manual Service BLE stack event handler.
 *
 * @details Handles all events from the BLE stack of interest to the mode
manual Service.
 *
 * @param[in]   p_mms     mode manual Service structure.
 * @param[in]   p_ble_evt  Event received from the BLE stack.
 */
void ble_mms_on_ble_evt(ble_mms_t * p_mms, ble_evt_t * p_ble_evt);


#endif // BLE_MMS_H__

/** @} */
```

```
/* Copyright (c) 2012 Nordic Semiconductor. All Rights Reserved.
 *
 * The information contained herein is property of Nordic Semiconductor ASA.
 * Terms and conditions of usage are described in detail in NORDIC
 * SEMICONDUCTOR STANDARD SOFTWARE LICENSE AGREEMENT.
 *
 * Licensees are granted free, non-transferable use of the information. NO
 * WARRANTY of ANY KIND is provided. This heading must NOT be removed from
 * the file.
 *
 */

#include "ble_mms.h"
#include <string.h>
#include "ble_srv_common.h"


#define INITIAL_OUTPUT 0


/**@brief mode manual event handler.
 *
 * @param[in]   p_mms       mode manual Service structure.
 * @param[in]   p_ble_evt   Event received from the BLE stack.
 */
static void on_mms(ble_mms_t * p_mms, ble_evt_t * p_ble_evt)
{
    ble_gatts_evt_write_t * p_evt_mms =
&p_ble_evt->evt.gatts_evt.params.write;

    if ((p_evt_mms->handle == p_mms->outPut_handles.value_handle) &&
(p_evt_mms->len == 1))
    {

if((p_evt_mms->data[0]==OUTPUT1_ON)||(p_evt_mms->data[0]==OUTPUT2_ON)||(p_ev
t_mms->data[0]==OUTPUT12_ON))
                    {
                    // Alert level written, call application event handler
                    ble_mms_evt_t evt;

            p_mms->mode  = MODE_MANUEL;
                    evt.evt_type = BLE_MMS_EVT_OUTPUT_UPDATED;

                    p_mms->outPut = p_evt_mms->data[0];
                    p_mms->evt_handler(p_mms, &evt);
                    }
            else if (p_evt_mms->data[0]==OUTPUT_OFF)
                    {
                    // Alert level written, call application event handler
                    ble_mms_evt_t evt;
          p_mms->mode  = MODE_MANUEL;
                    evt.evt_type = BLE_MMS_EVT_OUTPUT_UPDATED;
                    p_mms->outPut = p_evt_mms->data[0];
                    p_mms->evt_handler(p_mms, &evt);
                    }
        }

}
```

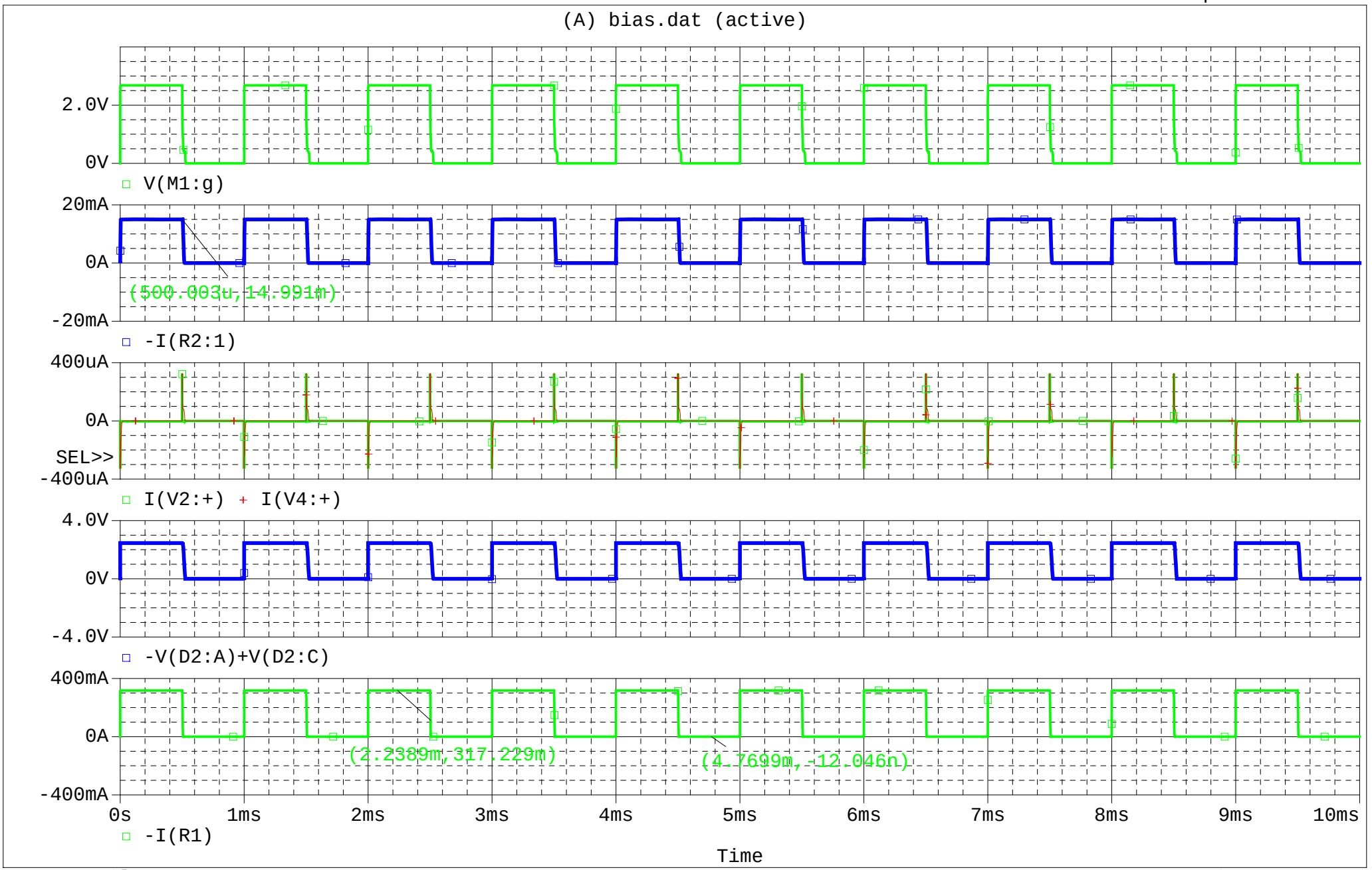


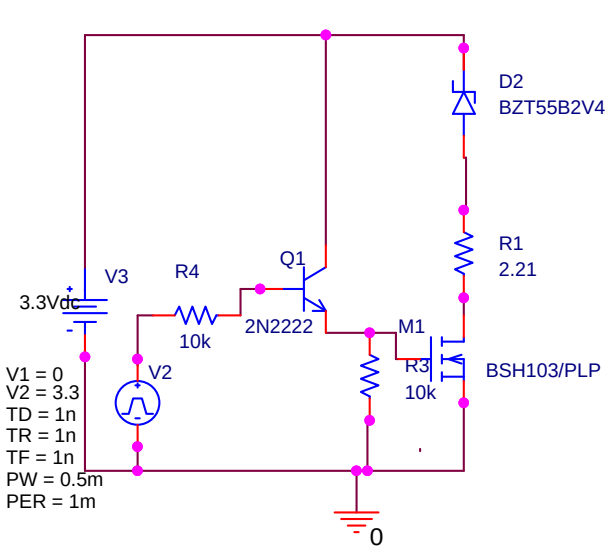
```
void ble_mms_on_ble_evt(ble_mms_t * p_mms, ble_evt_t * p_ble_evt)
{
    switch (p_ble_evt->header.evt_id)
    {
        case BLE_GATTS_EVT_WRITE:
            on_mms(p_mms, p_ble_evt);
            break;
        default:
            break;
    }
}


// /**@brief mms characteristic.
//  *
//  * @param[in]   p_mms       mms Service structure.
//  * @param[in]   p_mms_init   Information needed to initialize the
service.
//  *
//  * @return      NRF_SUCCESS on success, otherwise an error code.
//  */
static uint32_t outPut_char_add(ble_mms_t * p_mms)
{
    ble_gatts_char_md_t char_md;
    ble_gatts_attr_t    attr_char_value;
    ble_uuid_t          ble_uuid;
    ble_gatts_attr_md_t attr_md;
    uint8_t             initial_outPut;

    memset(&char_md, 0, sizeof(char_md));

    char_md.char_props.write_wo_resp = 1;
    char_md.p_char_user_desc         = NULL;
    char_md.p_char_pf                = NULL;
    char_md.p_user_desc_md           = NULL;
    char_md.p_cccd_md                = NULL;
    char_md.p_sccd_md                = NULL;

    BLE_UUID_BLE_ASSIGN(ble_uuid, BLE_UUID_MMS_SERVICE);

    memset(&attr_md, 0, sizeof(attr_md));

    BLE_GAP_CONN_SEC_MODE_SET_OPEN(&attr_md.read_perm);
    BLE_GAP_CONN_SEC_MODE_SET_OPEN(&attr_md.write_perm);
        //BLE_GAP_CONN_SEC_MODE_SET_ENC_NO_MITM(&attr_md.write_perm);

    attr_md.vloc       = BLE_GATTS_VLOC_STACK;
    attr_md.rd_auth    = 0;
    attr_md.wr_auth    = 0;
    attr_md.vlen       = 0;

    memset(&attr_char_value, 0, sizeof(attr_char_value));
    initial_outPut = INITIAL_OUTPUT;

    attr_char_value.p_uuid       = &ble_uuid;
    attr_char_value.p_attr_md    = &attr_md;
    attr_char_value.init_len     = sizeof(uint8_t);
    attr_char_value.init_offs    = 0;
    attr_char_value.max_len      = sizeof(uint8_t);
    attr_char_value.p_value      = &initial_outPut;
```

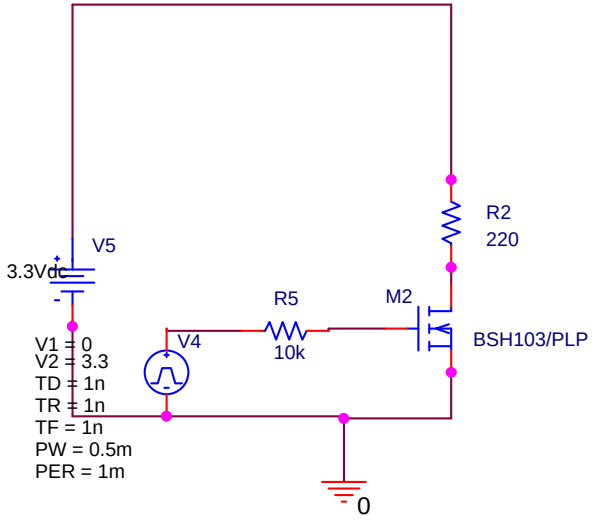
```
    return sd_ble_gatts_characteristic_add(p_mms->service_handle,
                                           &char_md,
                                           &attr_char_value,
                                           &p_mms->outPut_handles);
}


uint32_t ble_mms_init(ble_mms_t * p_mms, const ble_mms_init_t * p_mms_init)
{
    uint32_t   err_code;
    ble_uuid_t ble_uuid;

    // Initialize service structure
    if (p_mms_init->evt_handler == NULL)
    {
        return NRF_ERROR_INVALID_PARAM;
    }
    p_mms->evt_handler = p_mms_init->evt_handler;

    // Add service
    BLE_UUID_BLE_ASSIGN(ble_uuid, BLE_UUID_ALERT_NOTIFICATION_SERVICE);

    err_code = sd_ble_gatts_service_add(BLE_GATTS_SRVC_TYPE_PRIMARY,
&ble_uuid, &p_mms->service_handle);
    if (err_code != NRF_SUCCESS)
    {
        return err_code;
    }

    return outPut_char_add(p_mms);
}
```

** Profile: "SCHEMATIC1-bias" [C:\Cadence\SPB_16.5\tools\capture\v2-pspicefiles\schematic1\bias.sim]

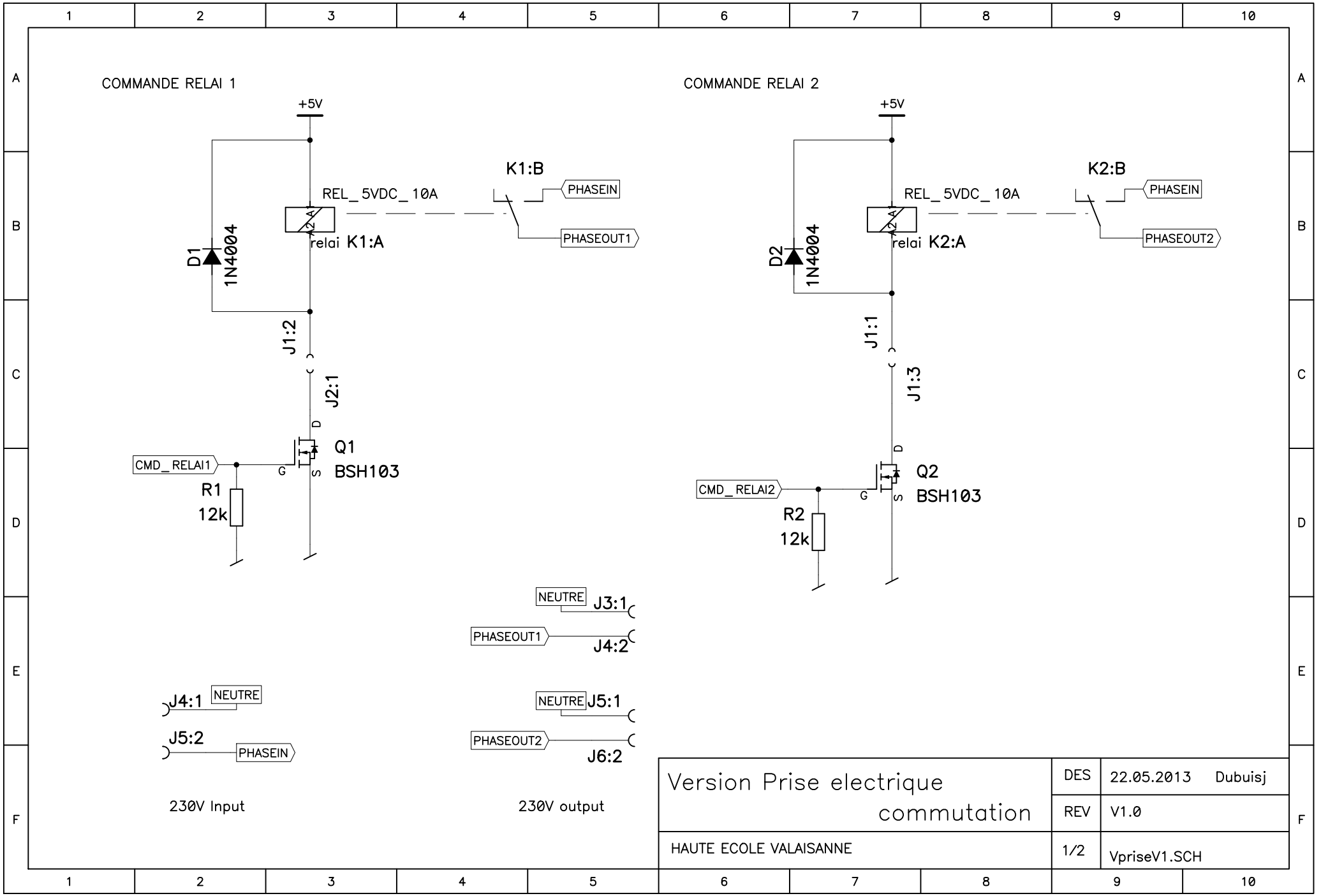
Date/Time run: 04/12/13 11:04:46 Temperature: 27.0



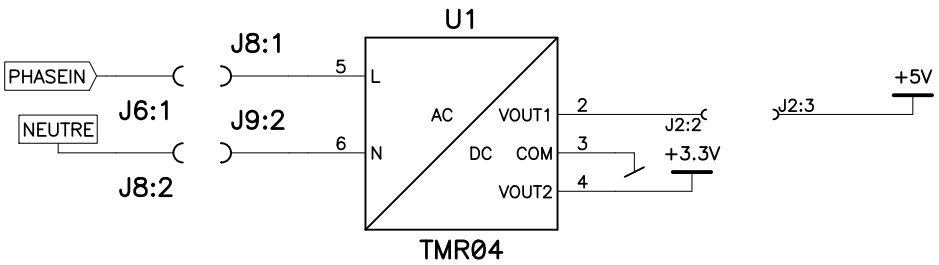




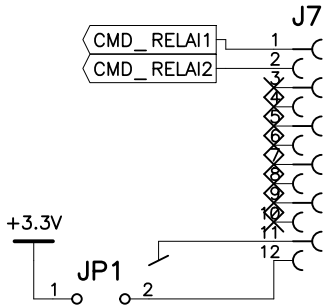
Title <Title>		
Size A	Document Number <Doc>	Rev <RevCode>
Date: Friday, April 12, 2013	Sheet 1 of 1	


COMMANDE RELAI 1
+5V
REL_5VDC_10A
K1:B
PHASEIN
PHASEOUT1
relai K1:A
D1
1N4004
J1:2
J2:1
Q1
BSH103
CMD_RELAI1
R1
12k
COMMANDE RELAI 2
+5V
REL_5VDC_10A
K2:B
PHASEIN
PHASEOUT2
relai K2:A
D2
1N4004
J1:1
J1:3
Q2
BSH103
CMD_RELAI2
R2
12k
NEUTRE
J3:1
PHASEOUT1
J4:2
J4:1
NEUTRE
J5:2
PHASEIN
NEUTRE
J5:1
PHASEOUT2
J6:2
230V Input
230V output
Version Prise electrique commutation
DES
22.05.2013 Dubuisj
REV
V1.0
HAUTE ECOLE VALAISANNE
1/2
VpriseV1.SCH

TRACO

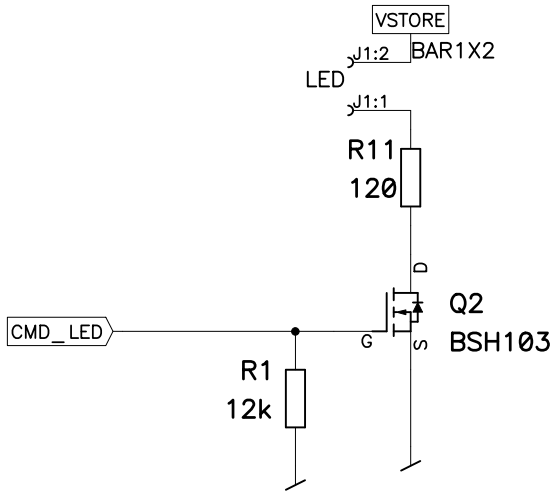


connector with the board nRF51822

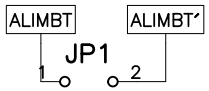


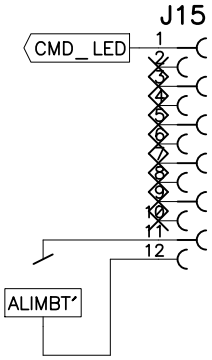
Version Prise electrique nRF51822	DES	22.05.2013	Dubuisj
	REV	V1.0	
HAUTE ECOLE VALAISANNE	2/2	VpriseV1.SCH	

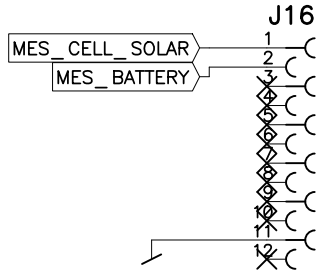
COMMANDE LED

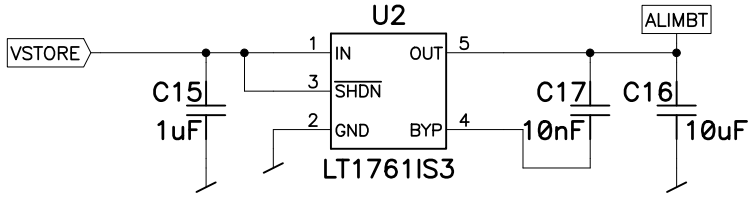


CONNECTION



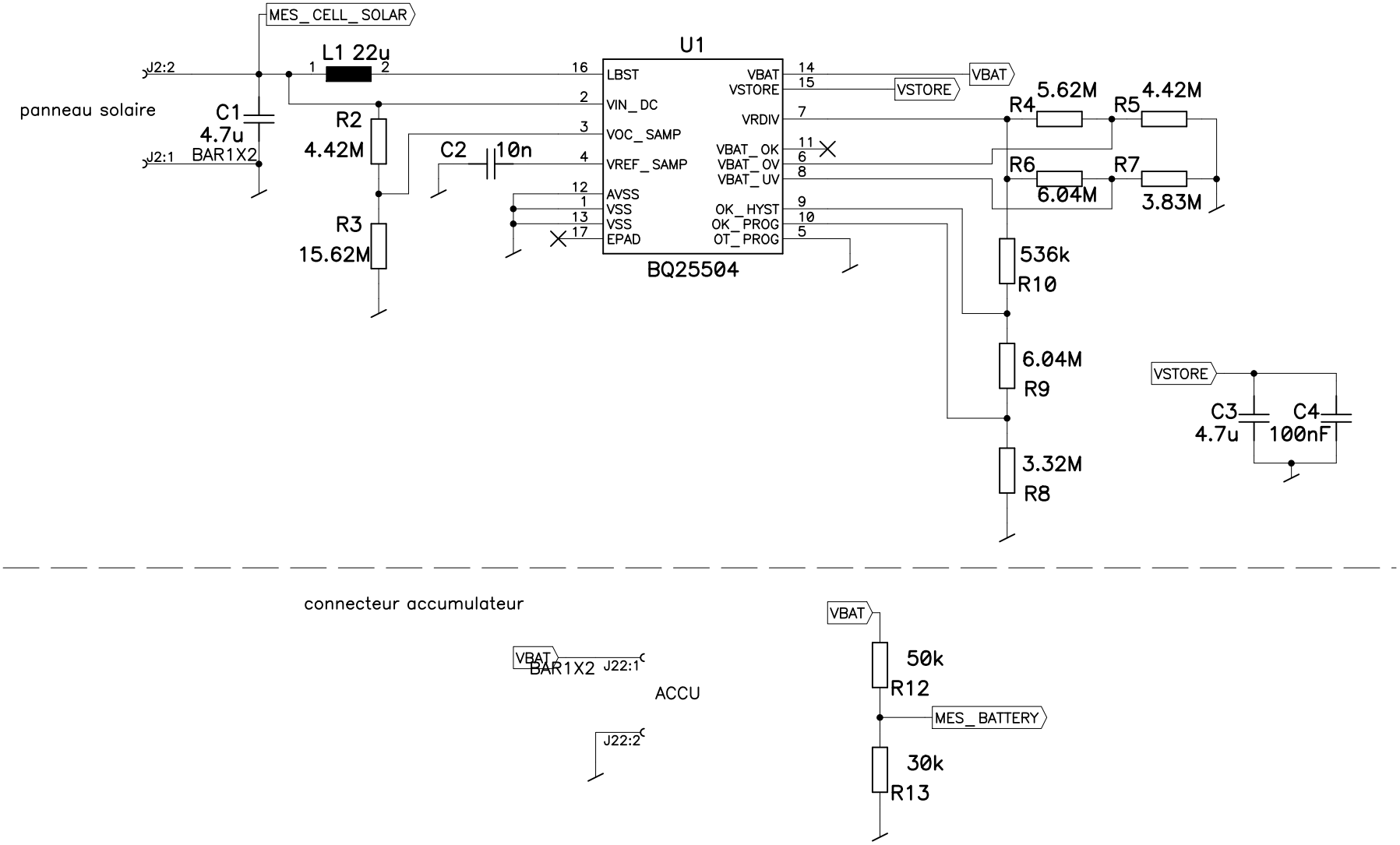






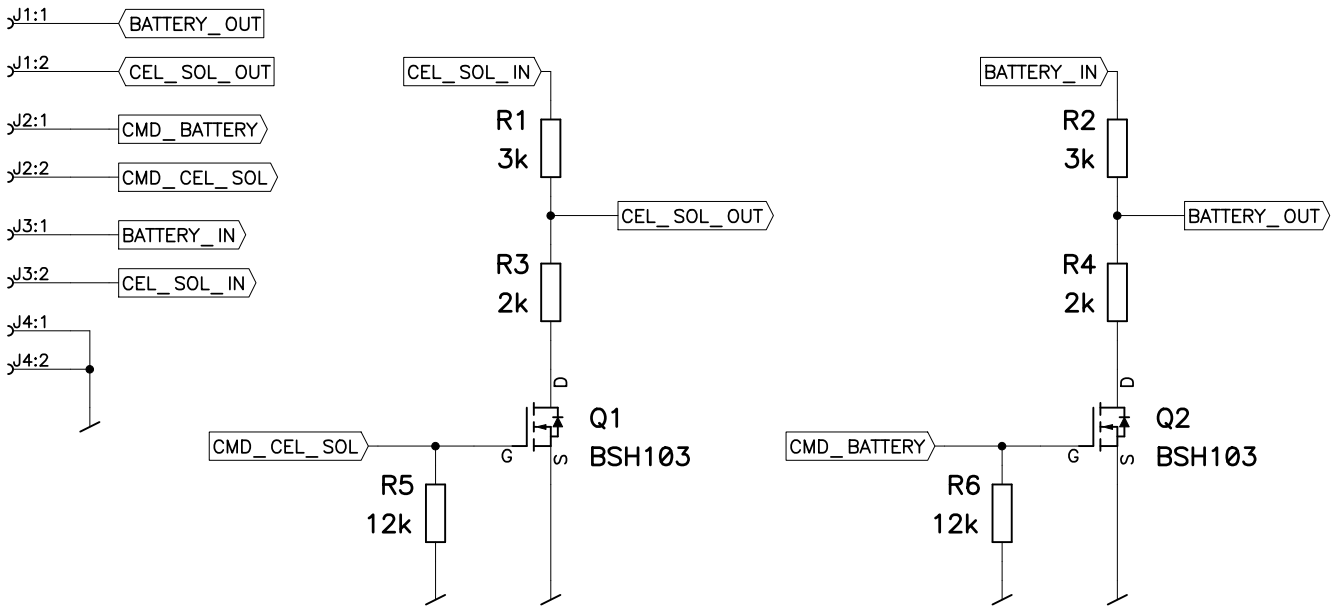
Lampe solaire commutation	DES	22.05.2013 dubuisj
	REV	V1.0
HAUTE ECOLE VALAISANNE	1/2	vsolaireV2.SCH

BQ25504

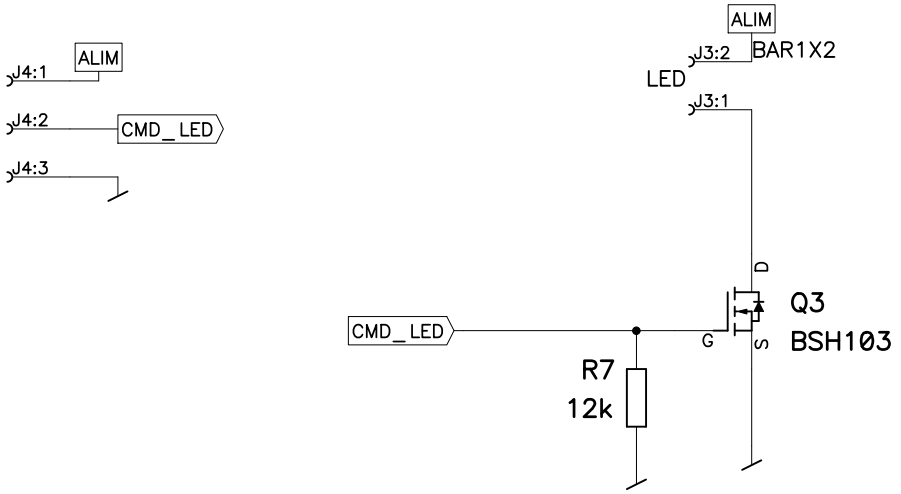


connecteur accumulateur

Lampe solaire Bq25504	DES	22.05.2013 dubuisj
	REV	V1.0
HAUTE ECOLE VALAISANNE	2/2	vsolaireV2.SCH

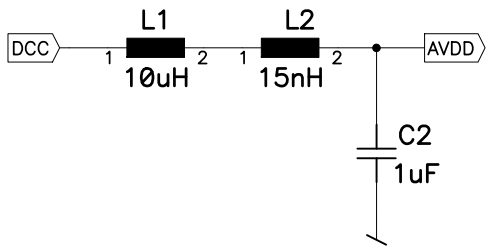


Lampe solaire adaptee mesure		DES	22.05.2013 dubuisj
		REV	V1.0
HAUTE ECOLE VALAISANNE		1/1	solaire_mesure.SCH

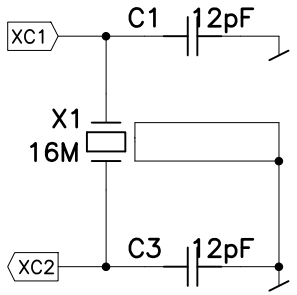


Lampe solaire adaptee Commande	DES	22.05.2013 dubuisj
	REV	V1.0
HAUTE ECOLE VALAISANNE	1/1	solaire_Mosfet.SCH

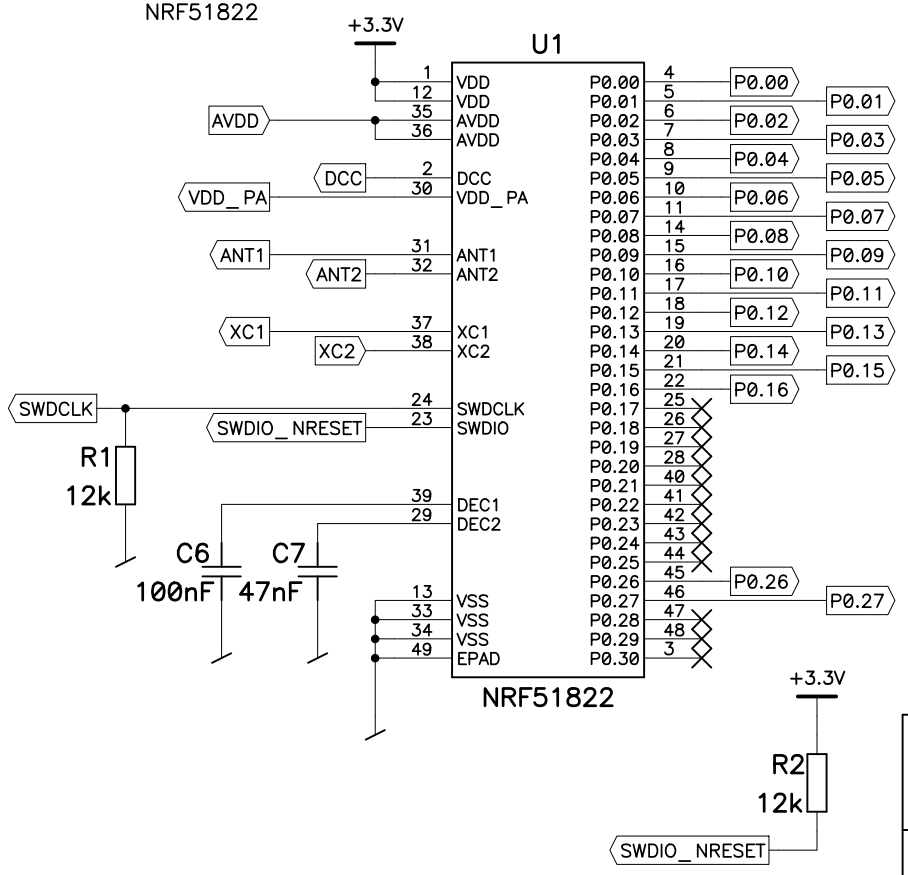
filtre low pass

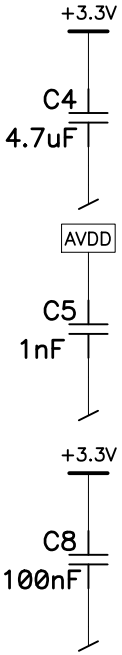


crystal 16M

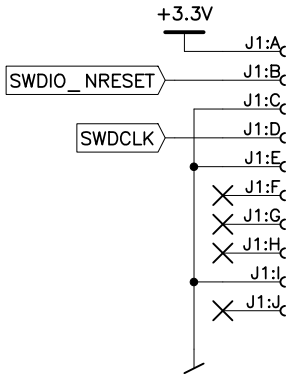


NRF51822



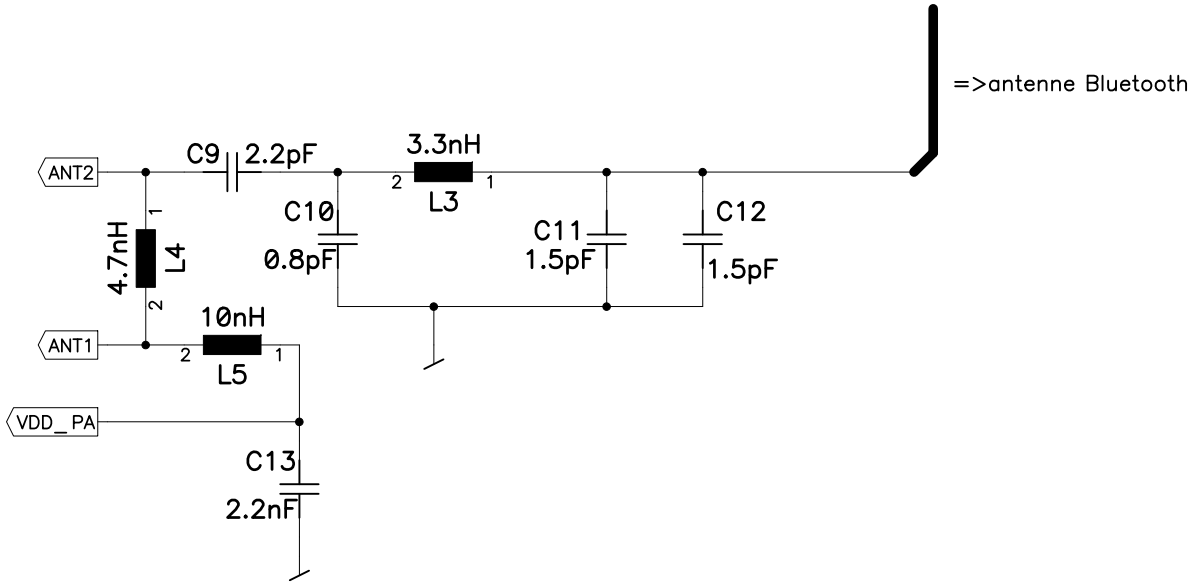


connector prog

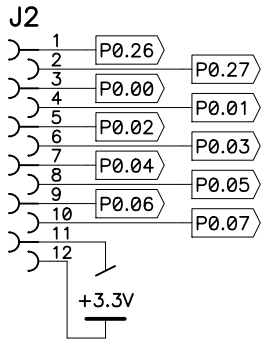


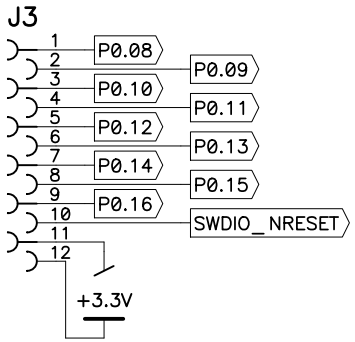
Board nRF51822 nRF51822	DES	22.05.2013 dubuisj
	REV	V1.0
HAUTE ECOLE VALAISANNE	1/4	Board_nRF51822.SCH

Antenne Bluetooth & adaptation



Connector





Board nRF51822 btAntenna	DES	22.05.2013 dubuisj
	REV	V1.0
HAUTE ECOLE VALAISANNE	2/4	Board_nRF51822.SCH